

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN
CƠ SỞ TRÍ TUỆ NHÂN TẠO



BÁO CÁO MÔN HỌC
PROJECT 01 – SEARCH ALGORITHMS

Giảng viên hướng dẫn : Lê Hoài Bắc
Lê Nhựt Nam

Nhóm thực hiện : Nhóm 9

Lớp : 23TNT1

Nhóm sinh viên thực hiện :

Bàng Mỹ Linh	23122009
Lại Nguyễn Hồng Thanh	23122018
Phan Huỳnh Châu Thịnh	23122019
Nguyễn Trọng Hòa	23122029

Tháng 11 năm 2025

Mục lục

1 MÔ TẢ THUẬT TOÁN	4
1.1 Các thuật toán Swarm Intelligence	4
1.1.1 Ant Colony Optimization (ACO)	4
1.1.2 Particle Swarm Optimization (PSO)	9
1.1.3 Artificial Bee Colony (ABC)	13
1.1.4 Firefly Algorithm (FA)	15
1.1.5 Cuckoo Search (CS)	18
1.2 Các thuật toán so sánh	20
1.2.1 A* Search	20
1.2.2 Genetic Algorithm (GA)	24
1.2.3 Simulated Annealing (SA)	26
2 TRIỂN KHAI VÀ THIẾT KẾ	29
2.1 Cấu trúc chương trình và môi trường	29
2.2 Thiết lập tham số	29
2.2.1 Các bài toán	32
3 PHƯƠNG PHÁP THỰC NGHIỆM	36
3.1 Phương pháp đánh giá	36
3.2 Visualize	37
3.2.1 Convergence	37
3.2.2 Comparative Performance	39
3.2.3 3D Surface	43
3.2.4 Parameter Sensitivity	46
4 KẾT QUẢ THỰC NGHIỆM VÀ PHÂN TÍCH	52
4.1 Problem 1: Traveling Salesman Problem (TSP)	52
4.1.1 Convergence Analysis	52
4.1.2 Comparative Performance	53
4.2 Problem 2: Sphere Function	58
4.2.1 Convergence Analysis	58
4.2.2 Parameter Sensitivity	61

4.2.3 Performance Metrics Comparison	65
4.3 Problem 3: Ackley Function	68
4.3.1 Convergence Analysis	68
4.3.2 Parameter Sensitivity	70
4.3.3 Performance Metrics Comparision	74
4.4 Problem 4: Rastrigin Function	79
4.4.1 Convergence Analysis	79
4.4.2 Parameter Sensitivity	80
4.4.3 Performance Metrics Comparison	83
4.5 Tổng quan thực nghiệm	87
4.5.1 Kết quả tổng hợp trên Continuous Problems	88
4.5.2 Kết quả trên Discrete Problem	90
4.5.3 Xếp hạng tổng thể	90
4.5.4 Scalability Analysis	91
4.5.5 Khuyến nghị lựa chọn thuật toán	92
5 KẾT LUẬN VÀ ĐÁNH GIÁ	93
5.1 Tổng kết đồ án	93
5.1.1 Mục tiêu và kết quả đạt được	93
5.2 Phát hiện chính	93
5.2.1 Về thuật toán	93
5.2.2 Về parameter tuning	94
5.3 Hạn chế và khó khăn	95
5.3.1 Hạn chế của đồ án	95
5.3.2 Khó khăn gặp phải	95
6 PHÂN CÔNG CÔNG VIỆC	96
6.1 *	97

1 MÔ TẢ THUẬT TOÁN

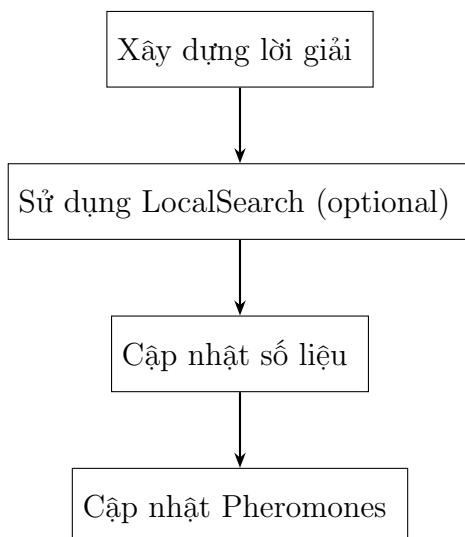
1.1 Các thuật toán Swarm Intelligence

1.1.1 Ant Colony Optimization (ACO)

Nguyên lý

Kỹ thuật tối ưu Ant Colony Optimization (ACO) được lấy cảm hứng từ hành vi tìm đường của loài kiến, dựa trên cơ chế pheromone, nhằm mục đích tìm một nghiệm tương đối tốt cho các bài toán tối ưu hóa đường đi (thường là NP-hard). Thuật toán cho ACO có nhiều biến thể, được phát triển trên nền thuật toán cơ bản nhất của ACO là Ant System (AS). Một vài trong số đó là: Max-Min Ant System (MMAS), Ant Colony System (ACS), Elitist Ant System (EAS) và Rank-based Ant System (ASRank).

Cấu trúc của một vòng lặp trong một thuật toán tối ưu ACO thường tương tự như sau:



1. Xây dựng lời giải:

Trong mỗi vòng lặp, mỗi cá thể kiến xuất phát từ một node ngẫu nhiên trong đồ thị, tại mỗi điểm lựa chọn, sẽ đưa ra quyết định một cách ngẫu nhiên. Mỗi lựa chọn có xác suất được chọn tùy theo lượng pheromone có trên mỗi lựa chọn cũng như chi phí của lựa chọn đó so với tổng thể tất cả các lựa chọn. Thông thường, một lựa chọn có lượng pheromone càng cao, chi phí càng thấp thì càng có khả năng được chọn.

2. Sử dụng LocalSearch (optional): Sau khi xây dựng lời giải cho đàn kiến, có thể sử dụng thuật toán Local Search nhằm cải thiện lời giải - thông thường, sẽ cải thiện lời giải tốt nhất trong vòng lặp đó (iteration-best).
3. Cập nhật số liệu: Đây là bước cập nhật các biến số liên quan cần thiết cho bước Cập nhật Pheromones và các biến số khác.

4. Cập nhật Pheromones: Thực hiện mô phỏng sự bay hơi của các pheromones cũng như sự bồi đắp pheromones trong quá trình di chuyển của kiến. Sự bay hơi pheromones cho phép quên dần đi các lời giải không tốt. Trong khi đó, sự bồi đắp pheromones giúp củng cố những lời giải tốt học được.

Công thức

Vì thuật toán AS là thuật toán cơ bản nhất của ACO, nhóm xin trình bày trước tiên cách thức hoạt động và các công thức liên quan của AS. Sau đó, lý giải lý do lựa chọn cũng như cách thức hoạt động và các công thức liên quan của thuật toán MMAS.

Công thức cập nhật pheromones của Ant System tại cuối vòng lặp $t - 1$, với mỗi path mà kiến k đi qua:

$$\tau_{ij}(t) = (1 - \rho) \cdot \tau_{ij}(t - 1) + \sum_{k=1}^m \Delta\tau_{ij}^k(t - 1)$$

với:

$$\Delta\tau_{ij}^k(t - 1) = \begin{cases} \frac{Q}{L_k} & \text{nếu kiến } k \text{ đã đi từ i sang j} \\ 0 & \text{nếu không} \end{cases}$$

Công thức xác định xác suất mỗi con kiến k lựa chọn đi từ node i sang node j:

$$p_{ij}^k(t) = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta} & \text{nếu } j \in \mathcal{N}_i^k \\ 0 & \text{nếu không} \end{cases}$$

Tuy nhiên, trong thực tế, thuật toán AS thua kém về khả năng tìm giải pháp tối ưu so với các biến thể cải tiến của nó. Trong số đó, MMAS luôn tạo ra những nghiệm chất lượng cao một cách ổn định [1], đặc biệt là đối với các bài toán có kích cỡ lớn [2]. Với lý do đó, nhóm đã chọn biến thể MMAS để thực hiện.

So với AS, MMAS khác ở hai đặc điểm mấu chốt:

- MMAS chỉ cho phép giải pháp tốt nhất tham gia vào quá trình bồi đắp pheromones (có thể là tốt nhất trong vòng lặp vừa chạy hoặc tốt nhất trong cả quá trình chạy của thuật toán). Trong khi, đối với AS, toàn bộ các giải pháp đều được tham gia vào quá trình này.
- MMAS giới hạn lượng pheromones trong khoảng $[\tau_{min}, \tau_{max}]$ nào đó. Công thức cập nhật pheromones của MMAS:

$$\tau_{ij}(t) = \left[(1 - \rho) \cdot \tau_{ij}(t - 1) + \sum_{k=1}^m \Delta\tau_{ij}^{\text{best}}(t - 1) \right]_{\tau_{min}}^{\tau_{max}}$$

với:

$$\Delta\tau_{ij}^k(t - 1) = \begin{cases} \frac{1}{L_{best}} & \text{nếu kiến best đã đi từ i sang j} \\ 0 & \text{nếu không} \end{cases}$$

Những đặc điểm này giúp MMAS cân bằng giữa hai khía cạnh then chốt trong ACO, đó là khai thác (exploitation) và khám phá (exploration). Điểm đầu tiên nhấn mạnh vào đặc

tính khai thác, khi chỉ tập trung cống cỗ những lời giải tốt nhất để thế hệ kiến ở vòng lặp tiếp theo tập trung vào những vùng hứa hẹn của không gian tìm kiếm. Tuy nhiên, việc tìm kiếm tham lam như vậy có nguy cơ gây ra hiện tượng hội tụ sớm, khi lời giải cuối cùng tương đối tốt nhưng không phải lời giải tối ưu. Điều này xảy ra khi một số đường đi tập trung một lượng quá cao pheromone, khiến thế hệ kiến sau gần như luôn đi theo những con đường đó mà không khám phá những khả năng khác, cùng lúc lại càng cống cỗ những đường đi này. Giải quyết yếu điểm này chính là vai trò của đặc điểm thứ hai. Việc không cho phép lượng pheromones vượt quá một ngưỡng trên hạn chế tạo sự khác biệt quá lớn giữa đường đi tốt nhất tìm được ở hiện tại so với các đường đi khác. Sự tồn tại của một ngưỡng dưới (lớn hơn 0) giúp cho xác suất một đường đi bất kỳ được chọn không bao giờ quá nhỏ hoặc bằng 0. Những điều này giúp khuyến khích đàn kiến khám phá nhiều hơn các đường đi khác, bù lại khuyết điểm sinh ra từ đặc điểm thứ nhất.

Ý nghĩa tham số

Bảng 1.1: Ý nghĩa các tham số trong ACO / MMAS

Tham số	Ký hiệu	Ý nghĩa
Kích thước quần thể	m	Số lượng kiến trong mỗi vòng lặp, ảnh hưởng đến mức độ đa dạng của các giải pháp được tạo ra.
Hệ số ảnh hưởng Pheromone	α	Điều khiển mức độ ảnh hưởng của đường mòn pheromone. Giá trị α lớn khiến kiến ưu tiên đi theo các đường mòn có nhiều pheromones.
Hệ số ảnh hưởng Heuristic	β	Điều khiển mức độ ảnh hưởng của thông tin heuristic. Giá trị β lớn khiến kiến ưu tiên các lựa chọn tốt về mặt cục bộ; mà trong bài toán TSP, nghĩa là kiến ưu tiên chọn những thành phố gần nhất.
Hệ số tồn đọng Pheromone	ρ	Tham số trong khoảng $[0, 1]$, kiểm soát tốc độ bay hơi. Giá trị ρ cao có nghĩa là pheromone tồn tại lâu hơn. $(1 - \rho)$ là tỷ lệ bay hơi.
Thông tin Heuristic	η_{ij}	Giá trị mong muốn cục bộ khi di chuyển từ i đến j . Trong bài toán TSP, nó thường được định nghĩa là $1/d_{ij}$, ưu tiên các cạnh ngắn hơn.
Chất lượng giải pháp tốt nhất	L_{best}	Trong bài toán TSP, thường là độ dài của hành trình ngắn nhất (có thể là iteration-best hoặc best-so-far) do một kiến tìm được. Giá trị này được dùng để tính toán lượng pheromone $\Delta\tau$ sẽ được bồi đắp.
Mức Pheromone tối đa	τ_{max}	Là đặc trưng riêng của MMAS. Giới hạn trên của lượng pheromone trên mỗi đường đi.
Mức Pheromone tối thiểu	τ_{min}	Là đặc trưng riêng của MMAS. Giới hạn dưới (lớn hơn 0) của lượng pheromone trên mỗi đường đi.

Chi tiết thuật toán

- **Bước 1: Khởi tạo (Hàm `__init__`)**

- Khởi tạo các tham số: số lượng kiến (n_{ants}), số vòng lặp tối đa (`max_iter`), hệ số ảnh hưởng pheromone α , hệ số ảnh hưởng heuristic β , và tốc độ bay hơi ρ .
- Lấy thông tin bài toán TSP: ma trận khoảng cách (`distance_matrix`) và ma trận heuristic η với $\eta_{ij} = 1/d_{ij}$.
- Tính toán giới hạn Pheromone:

1. Tìm một tour ban đầu C_{nn} (ví dụ: bằng giải thuật Nearest Neighbor – `_nearest_neighbor_tour_length`).

2. Tính giới hạn trên:

$$\tau_{\max} = \frac{1.0}{\rho \cdot C_{nn}}$$

(ký hiệu: `trail_max`).

3. Tính giới hạn dưới:

$$\tau_{\min} = \frac{\tau_{\max}}{2.0 \cdot \dim}$$

(ký hiệu: `trail_min`).

- Khởi tạo ma trận pheromone τ : gán tất cả các cạnh τ_{ij} giá trị τ_{\max} .
- Khởi tạo tour tốt nhất toàn cục: `best_so_far_length = infinity`.

- **Bước 2: Xây dựng lời giải (Hàm `_construct_solutions` và `_build_ant_tour`)**

- Bắt đầu vòng lặp chính (từ 0 đến `max_iter`).
- Cho mỗi con kiến k (từ 1 đến n_{ants}):
 1. Đặt kiến vào một thành phố xuất phát ngẫu nhiên.
 2. Lặp lại cho đến khi kiến đi đủ \dim thành phố:
 - * Kiến k ở thành phố i chọn thành phố j (chưa thăm) tiếp theo dựa trên **Quy tắc chuyển đổi trạng thái**:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \text{unvisited}} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}$$

- Tính toán và lưu lại độ dài tour (`_compute_tour_length`) cho mỗi con kiến.

- **Bước 3: Cập nhật thống kê (Hàm `_update_statistics`)**

- Xác định tour tốt nhất trong vòng lặp hiện tại (`iter_best_tour`, `iter_best_length`) từ các tour của n_{ants} .
- So sánh `iter_best_length` với `best_so_far_length`.
- Nếu `iter_best_length < best_so_far_length`:
 - * Cập nhật `best_so_far_length = iter_best_length` và lưu `best_so_far_tour`.
 - * **Cập nhật lại giới hạn Pheromone:** tính toán lại τ_{\max} và τ_{\min} dựa trên `best_so_far_length` mới.
- Cập nhật `restart_best_length` (tour tốt nhất kể từ lần khởi động lại gần nhất).

- **Bước 4: Bay hơi Pheromone (Hàm `_evaporate`)**

- Giảm lượng pheromone trên *tất cả* các cạnh (i, j) theo tốc độ bay hơi ρ :

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

- **Bước 5: Cập nhật Pheromone (Hàm `_mmas_pheromone_update` và `_deposit_pheromone`)**

- **Xác định tour cập nhật:** dựa trên logic của `u_gb` và `stagnation`, thuật toán chọn *một* tour (`tour_to_update`) để cộng pheromone. Tour này có thể là:

- * Tour tốt nhất vòng lặp (iter_best_tour),
- * Tour tốt nhất kể từ lần khởi động lại (restart_best_tour),
- * Tour tốt nhất toàn cục (best_so_far_tour) (nếu $u_gb = 1$ và bị trì trệ).
- **Cộng Pheromone:** chỉ các cạnh (i, j) nằm trong tour_to_update được cộng thêm pheromone:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}$$

với $\Delta\tau_{ij} = \frac{1}{L_{\text{update}}}$, trong đó L_{update} là độ dài của tour_to_update.

- **Bước 6: Kiểm soát giới hạn và Trì trệ (Hàm _check_pheromone_limits và _search_control)**

- **Ràng buộc Pheromone:** đảm bảo mọi giá trị τ_{ij} nằm trong khoảng $[\tau_{\min}, \tau_{\max}]$:
 - * Nếu $\tau_{ij} < \tau_{\min}$, gán $\tau_{ij} = \tau_{\min}$.
 - * Nếu $\tau_{ij} > \tau_{\max}$, gán $\tau_{ij} = \tau_{\max}$.
- **Kiểm soát trì trệ (Stagnation Control):** nếu thuật toán không cải thiện sau một số vòng lặp (ví dụ: iteration mod 100 = 0), thực hiện khởi động lại bằng cách đặt lại toàn bộ ma trận pheromone về τ_{\max} để thúc đẩy khám phá.
- **Bước 7: Kiểm tra điều kiện dừng (Hàm optimize)**
 - Nếu số vòng lặp hiện tại $\text{iteration} < \text{max_iter}$, quay lại Bước 2.
 - Ngược lại, thuật toán kết thúc và trả về best_so_far_tour cùng best_so_far_length.

1.1.2 Particle Swarm Optimization (PSO)

Nguyên lý

- Thuật toán được hình thành dựa trên việc mô phỏng hành vi xã hội của các sinh vật sống trong tự nhiên, đặc biệt là các bầy chim, đàn cá hoặc các nhóm động vật di chuyển tập thể. Nhiều nhà khoa học như Reynolds hay Heppner và Grenander đã xây dựng các mô phỏng máy tính thể hiện sự chuyển động đồng bộ của chim hoặc cá trong không gian.
- Trong khi Reynolds quan tâm đến tính thẩm mỹ của chuyển động bầy đàn, thì Heppner lại tập trung khám phá các quy tắc nền tảng cho phép số lượng lớn cá thể di chuyển đồng bộ, đổi hướng đột ngột, tách nhóm và hợp nhóm trở lại.
- Các mô hình này cho rằng sự đồng bộ đó đến từ việc các cá thể duy trì một khoảng cách tối ưu với những "láng giềng" của mình. Từ góc nhìn tiến hóa, việc chia sẻ thông tin giữa các cá thể trong bầy giúp nâng cao khả năng sinh tồn: khi một cá thể phát hiện được nguồn thức ăn hay điều kiện thuận lợi, thông tin này được lan truyền đến mỗi cá thể từ đó giúp cả bầy cùng có lợi. Điều này cũng áp dụng cho hành vi của con người trong xã hội, con người thường điều chỉnh hành vi, niềm tin và quyết định dựa trên những người xung quanh.
- Chính giả thuyết về "lợi ích của việc chia sẻ thông tin trong cộng đồng" là nền tảng để phát triển thuật toán PSO. Thay vì di chuyển trong không gian vật lý như chim hoặc cá, các cá thể trong PSO di chuyển trong không gian trừu tượng nhiều chiều

(không gian nghiệm) để tìm lời giải tối ưu. Mỗi cá thể vừa chịu ảnh hưởng từ kinh nghiệm của chính mình, vừa học hỏi từ những cá thể khác trong quần thể. Việc này cũng chính là đang mô phỏng quá trình học hỏi và thích nghi xã hội của con người.

Công thức

1. Cập nhật vị trí tốt nhất của mỗi cá thể: (cá thể thứ i)

$$p_{i,t+1}^d = \begin{cases} x_{i,t+1}^d, & \text{nếu } f(X_{i,t+1}) < f(P_{i,t}), \\ p_{i,t}^d, & \text{ngược lại.} \end{cases} \quad (1.1)$$

Trong đó:

- $p_{i,t+1}^d$: vị trí tốt nhất của cá thể tại thời điểm $t + 1$
- $x_{i,t+1}^d$: vị trí hiện tại của cá thể tại chiều d
- $p_{i,t}^d$: vị trí tốt nhất hiện tại của cá thể tại chiều d
- $P_{i,t+1}$:
- $X_{i,t} = (x_{i,t}^1, \dots, x_{i,t}^D)$: vector vị trí toàn phần của cá thể i tại thời điểm $t + 1$
- $P_{i,t} = (p_{i,t}^1, \dots, p_{i,t}^D)$: vector best cá nhân toàn phần của cá thể i tại thời điểm t
- $f(\cdot)$: hàm mục tiêu (fitness)

2. Cập nhật vị trí tốt nhất của cả bầy (global best):

$$p_{g,t+1}^d = \begin{cases} p_{i,t+1}^d, & \text{nếu } f(P_{i,t+1}) < f(P_{g,t}), \\ p_{g,t}^d, & \text{ngược lại.} \end{cases} \quad (1.2)$$

Trong đó:

- $p_{g,t+1}^d$: giá trị tốt nhất toàn cục tại chiều d sau khi cập nhật
- $p_{i,t+1}^d$: giá trị tốt nhất cá nhân của cá thể i tại chiều d ở thời điểm $t + 1$
- $p_{g,t}^d$: giá trị tốt nhất toàn cục của quần thể tại chiều d ở thời điểm t
- $P_{i,t+1} = (p_{i,t+1}^1, \dots, p_{i,t+1}^D)$: vector best cá nhân toàn phần của cá thể i tại thời điểm $t + 1$
- $P_{g,t} = (p_{g,t}^1, \dots, p_{g,t}^D)$: vector best toàn cục toàn phần của quần thể tại thời điểm t

3. Cập nhật vận tốc và vị trí của cá thể thứ i tại chiều d :

$$v_{i,t+1}^d = w v_{i,t}^d + c_1 r_1 (p_{i,t}^d - x_{i,t}^d) + c_2 r_2 (p_{g,t}^d - x_{i,t}^d) \quad (1.3)$$

$$x_{i,t+1}^d = x_{i,t}^d + v_{i,t+1}^d \quad (1.4)$$

Trong đó:

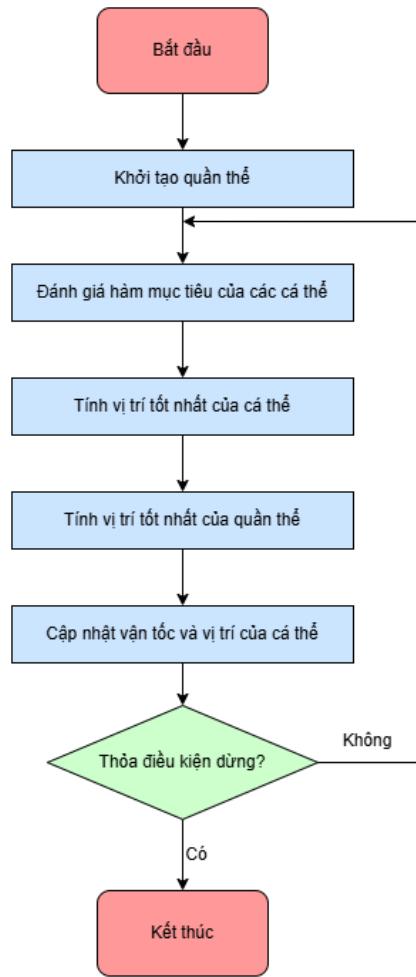
- $v_{i,t+1}^d$: vận tốc của cá thể i tại chiều d sau khi cập nhật
- $v_{i,t}^d$: vận tốc hiện tại
- $x_{i,t}^d$: vị trí hiện tại
- $x_{i,t+1}^d$: vị trí mới cập nhật
- $p_{i,t}^d$: giá trị tốt nhất cá nhân tại chiều d
- $p_{g,t}^d$: giá trị tốt nhất toàn cục tại chiều d

Ý nghĩa tham số

Bảng 1.2: Ý nghĩa các tham số trong PSO

Tham số	Ký hiệu	Ý nghĩa
Kích thước quần thể	N	Số lượng cá thể trong quần thể, ảnh hưởng đến khả năng khám phá không gian tìm kiếm.
Hệ số quán tính	w	Điều khiển mức độ ảnh hưởng của vận tốc cũ lên vận tốc mới; giá trị lớn \Rightarrow di chuyển xa, khám phá nhiều; giá trị nhỏ \Rightarrow tập trung quanh best.
Hệ số học tập cá nhân	c_1	Mức độ học từ chính bản thân (personal best).
Hệ số học tập xã hội	c_2	Mức độ học từ quần thể (global best).
Số ngẫu nhiên	r_1, r_2	Số ngẫu nhiên trong khoảng $[0,1]$, tạo tính stochastic trong cập nhật vận tốc, giúp tránh local optimum.

Chi tiết thuật toán



- Bước 1: Khởi tạo ngẫu nhiên một quần thể có N cá thể có các vị trí x_1, x_2, \dots, x_n và vận tốc v_1, v_2, \dots, v_n ban đầu.
- Bước 2: Tính giá trị hàm mục tiêu cho mỗi cá thể là $f(x_i)$. Khi đó quần thể sẽ có tập giá trị hàm mục tiêu: $f(x_1), f(x_2), \dots, f(x_n)$
- Bước 3: Tìm vị trí tốt nhất cho mỗi cá thể trong quần thể: $p_{i,t+1}$: So sánh giá trị hàm mục tiêu của vị trí hiện tại của cá thể $f(x_i)$ và vị trí tốt nhất hiện tại của cá thể $f(p_{i,t})$: Nếu như giá trị của cá thể hiện tại tốt hơn thì sẽ cập nhật vị trí tốt nhất chính là vị trí của cá thể tại thời điểm này, ngược lại giữ nguyên.
- Bước 4: Tìm vị trí tốt nhất của cả quần thể $p_{g,t+1}$: So sánh giá trị hàm mục tiêu của vị trí tốt nhất hiện tại của quần thể $f(p_g)$ với giá trị của các vị trí tốt nhất của các cá thể $f(p_i)$: Nếu như giá trị tốt nhất của cá thể tốt hơn thì cập nhật vị trí tốt nhất của cả quần thể là vị trí tốt nhất của cá thể này, ngược lại giữ nguyên.
- Bước 5: Cập nhật vận tốc v_i và vị trí x_i của các cá thể trong quần thể: Mỗi cá thể sẽ được điều chỉnh sao cho có hàm thích nghi với giá trị tốt nhất. Sau khi cập nhật thì sẽ kiểm tra xem vị trí và vận tốc có nằm trong khoảng giới hạn của bầy đàn hay không, nếu nhỏ hơn thì sẽ cập nhật bằng min của đàn, ngược lại lớn hơn thì sẽ cập nhật bằng max của đàn.

- Bước 6: Kiểm tra điều kiện dừng: Nếu thỏa thì sẽ kết thúc thuật toán, ngược lại thì lặp lại từ bước 2.

1.1.3 Artificial Bee Colony (ABC)

Nguyên lý

Thuật toán Bầy ong nhân tạo (Artificial Bee Colony - ABC) lấy cảm hứng từ hành vi tìm kiếm thức ăn thông minh của bầy ong mật. Về cơ bản, thuật toán này mô phỏng cách một đàn ong tìm kiếm, khai thác và chia sẻ thông tin về các nguồn mật hoa.

a. Phân chia lao động trong bầy ong:

Dàn ong nhân tạo trong thuật toán ABC được chia thành ba nhóm, tương ứng với các vai trò khác nhau của ong trong thế giới thực:

- Ong thợ (Employed bees): Những con ong này được liên kết với một nguồn thức ăn cụ thể. Chúng có nhiệm vụ tìm kiếm thức ăn xung quanh nguồn đã biết và thu thập thông tin về chất lượng (lượng mật) của nguồn đó.
- Ong quan sát (Onlooker bees): Những con ong này đợi trong tổ và nhận thông tin từ các ong thợ thông qua "điệu nhảy" của chúng. Dựa vào chất lượng của nguồn thức ăn mà ong thợ báo về, ong quan sát sẽ quyết định đi theo ong thợ nào để khai thác nguồn thức ăn đó. Những nguồn thức ăn chất lượng cao hơn sẽ thu hút nhiều ong quan sát hơn.
- Ong trinh sát (Scout bees): Khi một nguồn thức ăn bị cạn kiệt (giải pháp không được cải thiện sau một số lần thử), ong thợ tương ứng sẽ từ bỏ nguồn đó và trở thành một ong trinh sát. Ong trinh sát sẽ bay đi tìm kiếm một nguồn thức ăn hoàn toàn mới một cách ngẫu nhiên.

b. Quá trình tìm kiếm và khai thác mật hoa:

- Vị trí nguồn thức ăn tương ứng với giải pháp: Trong thuật toán, vị trí của một nguồn thức ăn trong không gian tìm kiếm đại diện cho một giải pháp khả thi của bài toán tối ưu.
- Lượng mật hoa tương ứng với độ tốt của giải pháp: Lượng mật hoa của một nguồn thức ăn tương ứng với “độ thích nghi” (fitness) hay chất lượng của giải pháp đó. Nguồn thức ăn càng có nhiều mật hoa thì giải pháp tương ứng càng tốt.
- Tìm kiếm cục bộ và toàn cục:
 - Khai thác: Ong thợ và ong quan sát thực hiện quá trình tìm kiếm cục bộ xung quanh các nguồn thức ăn đã biết. Điều này tương đương với việc tinh chỉnh và cải thiện các giải pháp hiện có.
 - Khám phá: Ong trinh sát thực hiện quá trình tìm kiếm toàn cục bằng cách khám phá những khu vực hoàn toàn mới trong không gian tìm kiếm. Điều này giúp thuật toán thoát khỏi các điểm tối ưu cục bộ và có cơ hội tìm ra điểm tối ưu toàn cục tốt hơn.

Trong một tổ ong thực, ong thợ sau khi tìm thấy nguồn thức ăn sẽ quay về tổ và thực hiện một “điệu nhảy” (waggle dance). Điệu nhảy này cung cấp thông tin về hướng, khoảng cách và chất lượng của nguồn thức ăn cho các con ong khác. Trong thuật toán ABC, quá trình này được mô phỏng bằng cách các ong thợ chia sẻ thông tin về độ tốt (fitness) của các giải pháp mà chúng đã tìm thấy. Dựa trên thông tin này, các ong quan sát sẽ lựa chọn những giải pháp có triển vọng nhất để tiếp tục khai thác.

Công thức

a. Chi tiết thuật toán: Chu trình của thuật toán ABC gồm 3 bước chính sau:

1. Khởi tạo:

- Tạo ra một quần thể ban đầu gồm SN nguồn thức ăn (tương ứng với SN nghiệm ứng viên của bài toán tối ưu). SN này đúng bằng số lượng của quần thể ong được khởi tạo.
- Trong số đó, một nửa quần thể là ong thợ và nửa còn lại là ong quan sát.
- Mỗi nguồn thức ăn x_i ($i = 1, 2, \dots, SN$) là một vector D -chiều. D chính là số lượng tham số cần tối ưu

2. Vòng lặp chính: Quá trình tìm kiếm được lặp lại như sau:

- *Pha Ong thợ:* Mỗi ong thợ bay đến một vị trí nguồn thức ăn mới trong vùng lân cận của vị trí hiện tại của nó. Nó đánh giá lượng mật (độ thích nghi) của nguồn mới thông qua hàm một hàm fitness và ghi nhớ vị trí có fitness cao nhất (vị trí tốt nhất trong lân cận nguồn thức ăn đó)
- *Pha Ong quan sát:* Mỗi ong quan sát sẽ chọn một nguồn thức ăn dựa trên xác suất tỷ lệ thuận với chất lượng của nguồn đó. Sau khi chọn xong, ong quan sát cũng tạo ra một vị trí mới trong vùng lân cận của nguồn đã chọn và áp dụng quy tắc lựa chọn tương tự như ong thợ.
- *Pha Ong trinh sát:* Nếu nguồn thức ăn không cải thiện sau một loạt chu kỳ khai thác, nó bị xem là cạn kiệt. Khi đó ong thợ sẽ biến thành ong trinh sát, rồi bỏ nguồn đó và đi tìm nguồn thức ăn mới hoàn toàn ngẫu nhiên.

3. Kết thúc: Quá trình lặp lại cho đến khi đạt điều kiện dừng (ví dụ: đạt số chu kỳ tối đa - MCN)

b. Các công thức toán học được sử dụng

1. Lựa chọn của Ong quan sát: Một con ong quan sát chọn một nguồn thức ăn i với xác suất p_i được tính như sau:

$$p_i = \frac{fit_i}{\sum_{i=1}^{SN} (fit_i)}$$

Trong đó:

- fit_i là giá trị độ thích nghi (chất lượng) của giải pháp i .
- SN là tổng số nguồn thức ăn (bằng số lượng ong thợ).

2. Tạo giải pháp ứng viên mới: Để tạo ra một vị trí nguồn thức ăn ứng viên mới (v_{ij}) từ vị trí cũ (x_{ij}), thuật toán ABC sử dụng công thức sau:

$$v_{ij} = x_{ij} + \Phi_{ij} \cdot (x_{ij} - x_{kj})$$

Trong đó:

- k là một chỉ số được chọn ngẫu nhiên từ $1, 2, \dots, SN$ và k phải khác i .
- j là một chỉ số chiều được chọn ngẫu nhiên từ $1, 2, \dots, D$ (D là số chiều của bài toán).
- Φ_{ij} là một số ngẫu nhiên trong đoạn $[-1, 1]$

Công thức này cho thấy rằng khi thuật toán tiến gần đến giải pháp tối ưu, sự khác biệt giữa x_{ij} và x_{kj} giảm xuống, làm cho bước nhảy (sự thay đổi) nhỏ lại, giúp thuật toán hội tụ tinh hơn.

Ý nghĩa tham số

Tham số	Ký hiệu	Giá trị	Ý nghĩa
Kích thước quần thể	SN	125	Tổng số giải pháp trong quần thể. Số ong thợ và ong quan sát mỗi loại chiếm 50% quần thể.
Giới hạn từ bỏ	limit	20	Số chu kỳ tối đa mà một giải pháp không được cải thiện.
Số chu kỳ tối đa	MCN	100, 500, 750, 1000,...	Điều kiện dừng của thuật toán, tương đương với số chu kỳ tối đa.

1.1.4 Firefly Algorithm (FA)

Nguyên lý

Thuật toán Đom đóm (Firefly Algorithm - FA) là một thuật toán metaheuristic lấy cảm hứng từ hành vi nhấp nháy và sự tương tác xã hội của loài đom đóm. Về cơ bản, thuật toán mô phỏng cách các con đom đóm bị thu hút bởi nhau dựa trên cường độ ánh sáng mà chúng phát ra, tương ứng với chất lượng của một giải pháp trong không gian tìm kiếm.

a. Hành vi và các quy tắc lý tưởng hóa:

Để đơn giản hóa và mô hình hóa thuật toán, FA dựa trên ba quy tắc lý tưởng hóa về hành vi của đom đóm:

- Đơn tính: Tất cả các con đom đóm được xem là đơn tính, nghĩa là bất kỳ con đom đóm nào cũng có thể bị thu hút bởi bất kỳ con đom đóm nào khác bất kể giới tính.
- Độ hấp dẫn và độ sáng: Độ hấp dẫn của một con đom đóm tỷ lệ thuận với độ sáng của nó. Do đó, đối với bất kỳ cặp đom đóm nào, con ít sáng hơn sẽ di chuyển về phía con sáng hơn. Độ hấp dẫn giảm khi khoảng cách giữa chúng tăng lên. Nếu không có con đom đóm nào sáng hơn một con cụ thể, nó sẽ di chuyển một cách ngẫu nhiên.

- Hàm mục tiêu và độ sáng: Độ sáng của một con đom đóm được xác định bởi cảnh quan của hàm mục tiêu. Dối với bài toán tối đa hóa, độ sáng có thể tỷ lệ thuận trực tiếp với giá trị của hàm mục tiêu.

b. Cường độ sáng và Độ hấp dẫn:

- Cường độ sáng tương ứng với độ tốt của giải pháp: Trong thuật toán, độ sáng của một con đom đóm tại một vị trí cụ thể trong không gian tìm kiếm đại diện cho chất lượng (độ thích nghi) của giải pháp đó. Một giải pháp tốt hơn sẽ tương ứng với một con đom đóm sáng hơn.
- Độ hấp dẫn phụ thuộc vào khoảng cách: Ánh sáng bị hấp thụ bởi môi trường, do đó cường độ sáng và độ hấp dẫn sẽ giảm theo khoảng cách. Một con đom đóm ở xa sẽ trông mờ hơn và do đó kém hấp dẫn hơn.
- Di chuyển dựa trên sự hấp dẫn: Quá trình tìm kiếm được thực hiện bằng cách các con đom đóm di chuyển về phía những con sáng hơn (tức là các giải pháp tốt hơn). Sự kết hợp giữa việc di chuyển có hướng tới các giải pháp tốt và một thành phần di chuyển ngẫu nhiên giúp cân bằng giữa việc khai thác và khám phá.

Công thức

a. Chi tiết thuật toán: Chu trình của thuật toán FA được mô tả trong pseudo-code như sau:

1. Khởi tạo:

- Xác định hàm mục tiêu $f(\mathbf{x})$.
- Tạo một quần thể ban đầu gồm n con đom đóm \mathbf{x}_i ($i = 1, 2, \dots, n$).
- Cường độ sáng I_i tại vị trí \mathbf{x}_i được xác định bởi $f(\mathbf{x}_i)$.
- Xác định hệ số hấp thụ ánh sáng γ .

2. Vòng lặp chính: Quá trình được lặp lại cho đến khi đạt điều kiện dừng (ví dụ: $t < \text{MaxGeneration}$):

- *So sánh và di chuyển:* Với mỗi con đom đóm i , so sánh nó với tất cả các con đom đóm j khác.
- *Di chuyển:* Nếu con đom đóm j sáng hơn con đom đóm i ($I_j > I_i$), con đom đóm i sẽ di chuyển về phía con đom đóm j .
- *Dánh giá và cập nhật:* Dánh giá các vị trí mới và cập nhật lại cường độ sáng của quần thể.

3. Kết thúc: Xếp hạng các con đom đóm và tìm ra giải pháp tốt nhất toàn cục hiện tại. Quá trình lặp lại cho đến khi thỏa mãn điều kiện dừng.

b. Các công thức toán học được sử dụng

1. Độ hấp dẫn (Attractiveness): Độ hấp dẫn β của một con đom đóm thay đổi theo khoảng cách r và được mô hình hóa theo công thức dạng Gaussian:

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

Trong đó:

- β_0 là độ hấp dẫn tại khoảng cách $r = 0$ (độ hấp dẫn gốc).
 - γ là hệ số hấp thụ ánh sáng của môi trường.
 - r là khoảng cách giữa hai con đom đóm.
2. Khoảng cách: Khoảng cách Descartes giữa hai con đom đóm i và j tại các vị trí \mathbf{x}_i và \mathbf{x}_j được tính bằng:

$$r_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\| = \sqrt{\sum_{k=1}^d (x_{i,k} - x_{j,k})^2}$$

3. Công thức di chuyển: Sự di chuyển của con đom đóm i bị thu hút bởi con đom đóm sáng hơn j được xác định bởi:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \beta_0 e^{-\gamma r_{ij}^2} (\mathbf{x}_j^t - \mathbf{x}_i^t) + \alpha \boldsymbol{\epsilon}_i$$

Trong đó:

- Thành phần thứ hai là do sự hấp dẫn.
- Thành phần thứ ba là một bước di chuyển ngẫu nhiên, với α là tham số ngẫu nhiên hóa và $\boldsymbol{\epsilon}_i$ là một vector các số ngẫu nhiên.

Ý nghĩa tham số

Tham số	Ký hiệu	Giá trị	Ý nghĩa
Số lượng đom đóm	n	20 - 50	Kích thước quần thể, số lượng giải pháp trong mỗi thế hệ.
Độ hấp dẫn gốc	β_0	1.0	Giá trị độ hấp dẫn khi khoảng cách giữa hai con đom đóm bằng không.
Hệ số hấp thụ	γ	0.1 - 10.0	Kiểm soát sự suy giảm của độ hấp dẫn theo khoảng cách. Tham số này rất quan trọng đối với tốc độ hội tụ.
Tham số ngẫu nhiên hóa	α	0.1 - 1.0	Kiểm soát độ lớn của thành phần di chuyển ngẫu nhiên, giúp thuật toán thoát khỏi các điểm tối ưu cục bộ.
Số thế hệ tối đa	MaxGeneration	100 - 1000	Điều kiện dừng của thuật toán, tương đương với số vòng lặp tối đa.

1.1.5 Cuckoo Search (CS)

Nguyên lý

Thuật toán Cuckoo Search (CS) được phát triển bởi Xin-She Yang và Suash Deb vào năm 2009, lấy cảm hứng từ hành vi sinh sản ký sinh của chim cuckoo và kết hợp với lý thuyết chuyển động Lévy flights.

Hành vi sinh sản ký sinh: Một số loài chim cuckoo có thói quen đặt trứng vào tổ của các loài chim khác (brood parasitism). Chim chủ nhà có thể phát hiện trứng lạ và vứt bỏ, hoặc bỏ tổ và xây tổ mới. Hành vi này tạo nên cơ chế lựa chọn tự nhiên, trong đó chỉ những "nghiệm tốt" (trứng không bị phát hiện) mới được giữ lại.

Lévy Flights: Là dạng random walk với độ dài bước nhảy tuân theo phân phối Lévy. Đây là chiến lược tìm kiếm hiệu quả được quan sát thấy ở nhiều động vật săn mồi (cá mập, ong mật, chim). Lévy flights cho phép thuật toán thực hiện:

- **Bước nhảy ngắn:** Khai thác (exploitation) vùng xung quanh nghiệm tốt
- **Bước nhảy dài:** Khám phá (exploration) các vùng mới trong không gian tìm kiếm

Ba quy tắc lý tưởng hóa của CS:

1. Mỗi con cuckoo đặt một quả trứng một lúc và đặt vào tổ được chọn ngẫu nhiên
2. Tổ tốt nhất (có chất lượng cao nhất) được giữ lại cho thế hệ tiếp theo
3. Số lượng tổ chủ nhà là cố định, xác suất phát hiện trứng lạ là $p_a \in [0, 1]$

Công thức

1. Cập nhật vị trí nghiệm bằng Lévy Flights:

$$x_i^{(t+1)} = x_i^t + \alpha \oplus \text{Lévy}(\lambda) \quad (1.5)$$

Trong đó:

- x_i^t : vị trí tổ thứ i tại thời điểm t
- α : hệ số tỷ lệ bước nhảy (step size scaling factor)
- \oplus : phép nhân từng phần tử (element-wise multiplication)
- $\text{Lévy}(\lambda)$: vector ngẫu nhiên tuân theo phân phối Lévy

2. Phân phối Lévy (Mantegna's method):

$$s = \frac{u}{|v|^{1/\beta}} \quad (1.6)$$

Với:

$$u \sim \mathcal{N}(0, \sigma_u^2) \quad (1.7)$$

$$v \sim \mathcal{N}(0, \sigma_v^2 = 1) \quad (1.8)$$

$$\sigma_u = \left[\frac{\Gamma(1 + \beta) \times \sin(\pi\beta/2)}{\Gamma((1 + \beta)/2) \times \beta \times 2^{(\beta-1)/2}} \right]^{1/\beta} \quad (1.9)$$

Trong đó β là hệ số mũ Lévy (thường $\beta = 1.5$) và Γ là hàm Gamma.

3. Cơ chế loại bỏ nghiệm xấu:

Với xác suất p_a (discovery rate), nghiệm xấu bị thay thế:

$$x_i^{(t+1)} = x_i^t + r \times (x_j^t - x_k^t) \quad (1.10)$$

Với $r \sim U(0, 1)$ và x_j, x_k là hai nghiệm được chọn ngẫu nhiên.

Ý nghĩa tham số

Tham số	Ký hiệu	Giá trị	Ý nghĩa
Số lượng tổ	n_{nests}	25	Kích thước quần thể
Xác suất phát hiện	p_a	0.25	Tỷ lệ nghiệm bị loại bỏ
Hệ số Lévy	β	1.5	Mũ phân phối Lévy
Hệ số bước nhảy	α	0.01-0.1	Độ lớn bước di chuyển

Bảng 1.3: Các tham số chính của Cuckoo Search

Ảnh hưởng của tham số:

- n_{nests} : Lớn \rightarrow khám phá tốt nhưng chậm; Nhỏ \rightarrow nhanh nhưng dễ kẹt
- p_a : Cao (≥ 0.3) \rightarrow nhiều exploration; Thấp (≤ 0.2) \rightarrow nhiều exploitation
- β : Giá trị tối ưu $\beta = 1.5$ theo lý thuyết, ít ảnh hưởng khi thay đổi
- α : Lớn (≥ 0.1) \rightarrow bước xa, khám phá rộng; Nhỏ (≤ 0.01) \rightarrow khai thác cục bộ

Chi tiết thuật toán

Thuật toán Cuckoo Search bao gồm các bước chính như sau:

Bước 1 - Khởi tạo: Tạo quần thể ban đầu gồm n tổ chim với các vị trí ngẫu nhiên trong không gian tìm kiếm. Mỗi tổ chim i được đánh giá chất lượng thông qua hàm mục tiêu $f(x_i)$. Nghiệm tốt nhất x_{best} và giá trị mục tiêu tương ứng f_{best} được ghi nhận.

Bước 2 - Tạo nghiệm mới bằng Lévy flights: Với mỗi tổ chim i , tạo nghiệm mới bằng cách thực hiện bước nhảy Lévy từ vị trí hiện tại:

$$x_i^{new} = x_i + \alpha \cdot L(\beta)$$

trong đó $L(\beta)$ là phân phối Lévy với tham số β , và α là hệ số bước nhảy. Nghiệm mới được giới hạn trong miền khả thi.

Bước 3 - Lựa chọn tham lam: Đánh giá nghiệm mới x_i^{new} . Chọn ngẫu nhiên một tổ chim j khác trong quần thể. Nếu nghiệm mới tốt hơn nghiệm tại tổ j (tức là $f(x_i^{new}) < f(x_j)$), thay thế nghiệm tại tổ j bằng nghiệm mới.

Bước 4 - Loại bỏ nghiệm xấu: Với xác suất phát hiện p_a , một tỷ lệ các nghiệm xấu nhất trong quần thể sẽ bị loại bỏ. Cụ thể, $\lfloor p_a \cdot n \rfloor$ nghiệm có giá trị mục tiêu lớn nhất được thay thế bằng nghiệm mới. Nghiệm mới được tạo dựa trên sự khác biệt giữa hai nghiệm ngẫu nhiên j và k trong quần thể:

$$x_i = x_i + r \cdot (x_j - x_k)$$

với r là số ngẫu nhiên trong khoảng $[0, 1]$.

Bước 5 - Cập nhật nghiệm tốt nhất: So sánh tất cả các nghiệm trong quần thể hiện tại. Nếu tìm thấy nghiệm tốt hơn x_{best} , cập nhật nghiệm tốt nhất và giá trị mục tiêu tương ứng.

Bước 6 - Kiểm tra điều kiện dừng: Lặp lại các bước 2-5 cho đến khi đạt số vòng lặp tối đa hoặc thỏa mãn điều kiện hội tụ. Trả về nghiệm tốt nhất x_{best} và giá trị mục tiêu f_{best} .

1.2 Các thuật toán so sánh

1.2.1 A* Search

Giới thiệu

A* (A-star) là một thuật toán tìm kiếm có thông tin (informed search) được sử dụng để tìm đường đi ngắn nhất trong đồ thị. Đối với bài toán TSP, A* khám phá không gian trạng thái bằng cách sử dụng hàm đánh giá:

$$f(n) = g(n) + h(n) \quad (1.11)$$

trong đó:

- $g(n)$: chi phí thực tế từ trạng thái ban đầu đến trạng thái n
- $h(n)$: chi phí ước lượng (heuristic) từ trạng thái n đến đích
- $f(n)$: tổng chi phí ước lượng của đường đi qua trạng thái n

Biểu diễn trạng thái

Mỗi trạng thái trong không gian tìm kiếm được định nghĩa bởi lớp `TSPState` gồm các thành phần:

- **path**: Danh sách các thành phố đã được thăm theo thứ tự
- **cost**: Tổng chi phí của đường đi hiện tại (giá trị $g(n)$)
- **remaining**: Tập hợp các thành phố chưa được thăm
- **current _ city**: Thành phố hiện tại (thành phố cuối cùng trong path)

Trạng thái ban đầu bắt đầu từ thành phố xuất phát với $cost = 0$ và `remaining` chứa tất cả các thành phố còn lại.

Ý nghĩa các tham số

Hàm Heuristic

Hàm heuristic đóng vai trò then chốt trong hiệu quả của thuật toán A*. Một heuristic tốt phải thỏa mãn hai điều kiện:

1. **Admissible**: Không bao giờ đánh giá quá cao chi phí thực tế (optimistic)
2. **Consistent**: $h(n) \leq c(n, n') + h(n')$ với mọi trạng thái n, n' kề nhau

Bảng 1.4: Các tham số chính của thuật toán A*

Tham số	Ý nghĩa
<code>problem</code>	Đối tượng bài toán TSP chứa ma trận khoảng cách và thông tin về các thành phố
<code>distance_matrix</code>	Ma trận $n \times n$ lưu khoảng cách giữa mọi cặp thành phố, với d_{ij} là khoảng cách từ thành phố i đến j
<code>n</code>	Số lượng thành phố trong bài toán
<code>time_limit</code>	Thời gian tối đa cho phép thuật toán chạy (mặc định 3600 giây = 1 giờ)
<code>start_city</code>	Thành phố xuất phát của tour (mặc định là thành phố 0)
<code>open_set</code>	Priority queue chứa các trạng thái đang chờ được khám phá, sắp xếp theo giá trị $f(n)$
<code>best_cost</code>	Chi phí của lời giải tốt nhất tìm được cho đến thời điểm hiện tại
<code>best_solution</code>	Tour hoàn chỉnh tốt nhất tìm được

MST Heuristic (Minimum Spanning Tree)

Trong implementation này, chúng ta sử dụng heuristic dựa trên cây khung nhỏ nhất (MST). Hàm `_mst_heuristic` tính toán chi phí ước lượng như sau:

$$h(n) = \text{MST_cost} + \min_{c \in R} d(\text{current}, c) + \min_{c \in R} d(c, \text{start}) \quad (1.12)$$

trong đó:

- `MST_cost`: Chi phí của cây khung nhỏ nhất trên tập thành phố chưa thăm R
- $\min_{c \in R} d(\text{current}, c)$: Cạnh ngắn nhất từ thành phố hiện tại đến một thành phố chưa thăm
- $\min_{c \in R} d(c, \text{start})$: Cạnh ngắn nhất từ một thành phố chưa thăm về thành phố xuất phát

Tính chất của MST Heuristic:

1. **Admissible**: MST luôn cho chi phí nhỏ hơn hoặc bằng Hamiltonian path tối ưu trên cùng tập đỉnh, vì Hamiltonian path có thể được chuyển thành spanning tree bằng cách bỏ một cạnh
2. **Informative**: MST cung cấp ước lượng chật chẽ hơn so với các heuristic đơn giản như "nearest neighbor" hay "zero heuristic"
3. **Computational Cost**: Việc tính MST có độ phức tạp $O(m^2 \log m)$ với m là số thành phố chưa thăm, sử dụng thuật toán Prim

Thuật toán Prim cho MST

Thuật toán Prim được thực hiện qua các bước chính:

1. Khởi tạo: Gán tất cả các key = ∞ , đánh dấu tất cả đỉnh chưa có trong MST.
2. Chọn đỉnh bắt đầu, gán key = 0.
3. Trong khi còn đỉnh chưa được thêm vào MST:
 - (a) Chọn đỉnh u có key nhỏ nhất chưa vào MST.
 - (b) Thêm u vào MST, cộng chi phí vào tổng.
 - (c) Cập nhật key của các đỉnh kề nếu cạnh nhỏ hơn key hiện tại.

Thuật toán A* cho TSP

Thuật toán A* được thực hiện để tìm đường đi ngắn nhất đi qua tất cả các thành phố (TSP) với các bước chính:

1. Khởi tạo trạng thái ban đầu $initial$:
 - $path = [start_city]$
 - $cost = 0$
 - $remaining = \{0, 1, \dots, n - 1\} \setminus \{start_city\}$
 - $f_cost = g(initial) + h(initial)$
2. Đưa $initial$ vào $open_set$, đặt $best_cost = \infty$, $best_solution = \text{None}$.
3. Trong khi $open_set \neq \emptyset$ và chưa hết thời gian:
 - (a) Chọn trạng thái $state$ có f_cost nhỏ nhất từ $open_set$.
 - (b) Nếu $f_cost \geq best_cost$, bỏ qua trạng thái (pruning).
 - (c) Nếu $state.remaining = \emptyset$ (đã thăm hết các thành phố):
 - Tính $complete_cost = state.cost + d(state.current, start)$.
 - Nếu $complete_cost < best_cost$, cập nhật $best_cost$ và $best_solution$.
 - Loại bỏ các trạng thái trong $open_set$ có $f \geq best_cost$.
 - (d) Với mỗi $next_city \in state.remaining$:
 - Tính chi phí mới $new_cost = state.cost + d(state.current, next_city)$.
 - Nếu $new_cost \geq best_cost$, bỏ qua.
 - Tạo trạng thái mới new_state với $next_city$.
 - Tính $new_f_cost = new_cost + h(new_state)$.
 - Nếu $new_f_cost < best_cost$, đưa new_state vào $open_set$.
4. Khi kết thúc, trả về tour tối ưu, chi phí, thời gian thực hiện, trạng thái tối ưu, và thông báo timeout nếu có.

Kỹ thuật tối ưu hóa

1. Pruning (Cắt tỉa nhánh):

- Loại bỏ các trạng thái có $f(n) \geq best_cost$
- Không mở rộng các trạng thái có $g(n) \geq best_cost$
- Cập nhật lại $open_set$ khi tìm được lời giải tốt hơn

2. State ID Counter: Dảm bảo thứ tự FIFO cho các trạng thái có cùng giá trị f , tránh lặp vô hạn

3. Time Limit: Giới hạn thời gian chạy để tránh các bài toán quá lớn chạy quá lâu

4. Early Termination: Dừng ngay khi tìm được lời giải tối ưu (khi $open_set$ rỗng)

Độ phức tạp

• Độ phức tạp thời gian:

- Worst case: $O(b^d)$ với b là branching factor ($\approx n$) và d là độ sâu ($= n$)
- Với MST heuristic: Mỗi trạng thái cần $O(m^2 \log m)$ để tính heuristic
- Tổng thể: Exponential trong trường hợp xấu nhất

• Độ phức tạp không gian: $O(b^d)$ để lưu trữ $open_set$ và các trạng thái

Ưu và nhược điểm

Ưu điểm:

- Dảm bảo tìm được lời giải tối ưu nếu heuristic admissible
- MST heuristic cung cấp ước lượng chặt chẽ, giúp cắt tỉa hiệu quả
- Có khả năng trả về lời giải tốt nhất hiện có khi hết thời gian

Nhược điểm:

- Tiêu tốn bộ nhớ lớn do phải lưu trữ toàn bộ $open_set$
- Thời gian chạy có thể rất lâu với bài toán lớn
- Chi phí tính toán heuristic cao (phải tính MST cho mỗi trạng thái)
- Không phù hợp với bài toán có số thành phố lớn ($> 20-30$ thành phố)

Kết quả trả về

Thuật toán trả về một dictionary chứa:

- tour: Tour hoàn chỉnh tốt nhất (list các thành phố)
- cost: Chi phí của tour
- time_elapsed: Thời gian thực thi
- optimal: Boolean cho biết lời giải có tối ưu hay không
- timed_out: Boolean cho biết có vượt quá thời gian cho phép không

1.2.2 Genetic Algorithm (GA)

Nguyên lý

Giải thuật Di truyền (Genetic Algorithm - GA) là một thuật toán tìm kiếm được lấy cảm hứng trực tiếp từ phép ẩn dụ về quá trình chọn lọc tự nhiên trong sinh học của Darwin. Về cơ bản, GA duy trì một quần thể các cá thể (tương ứng với các trạng thái hoặc giải pháp), trong đó những cá thể "khỏe mạnh" nhất (có giá trị cao nhất) sẽ tạo ra "con cái" (trạng thái kế thừa) để hình thành thế hệ tiếp theo thông qua một quá trình gọi là tái tổ hợp (recombination).

a. Các thành phần cốt lõi của thuật toán:

Một thuật toán di truyền được định nghĩa bởi các thành phần chính sau:

- Quần thể (Population): Một tập hợp các cá thể, mỗi cá thể đại diện cho một giải pháp khả thi cho bài toán.
- Cá thể (Individual): Một giải pháp đơn lẻ, thường được mã hóa dưới dạng một chuỗi trên một bảng chữ cái hữu hạn (ví dụ: chuỗi bit), tương tự như chuỗi DNA. Chuỗi này được gọi là nhiễm sắc thể (chromosome).
- Hàm thích nghi (Fitness Function): Một hàm đánh giá mức độ "tốt" của một cá thể (giải pháp). Các cá thể có điểm thích nghi cao hơn được coi là các giải pháp tốt hơn và có nhiều khả năng được chọn để sinh sản hơn.

b. Các toán tử di truyền chính:

Quá trình tiến hóa từ thế hệ này sang thế hệ tiếp theo được điều khiển bởi ba toán tử chính:

- Lựa chọn (Selection): Quá trình lựa chọn các cá thể từ quần thể hiện tại để trở thành "cha mẹ" cho thế hệ tiếp theo. Một phương pháp phổ biến là lựa chọn các cá thể với xác suất tỷ lệ thuận với điểm thích nghi của chúng. Điều này có nghĩa là những cá thể khỏe mạnh hơn có cơ hội sinh sản cao hơn.
- Lai ghép (Crossover): Đây là toán tử trung tâm của GA. Nó mô phỏng việc sinh sản hữu tính. Hai cá thể cha mẹ được chọn, và một "điểm lai ghép" (crossover point) được chọn ngẫu nhiên trên chuỗi nhiễm sắc thể của chúng. Các nhiễm sắc thể con được tạo ra bằng cách trao đổi các phân đoạn gen của cha mẹ tại điểm lai ghép đó. Điều này cho phép kết hợp các "khối xây dựng" (schemas) tốt từ các cá thể khác nhau.
- Đột biến (Mutation): Sau khi lai ghép, mỗi bit trong nhiễm sắc thể của con cái có một xác suất nhỏ bị thay đổi ngẫu nhiên (ví dụ: lật từ 0 thành 1). Đột biến giúp duy trì sự đa dạng di truyền trong quần thể và ngăn thuật toán hội tụ quá sớm vào một điểm tối ưu cục bộ.
- Thuyết tinh hoa (Elitism): Một kỹ thuật tùy chọn nhưng phổ biến, trong đó một vài cá thể có điểm thích nghi cao nhất từ thế hệ hiện tại được chuyển thẳng sang thế hệ tiếp theo mà không qua lai ghép hay đột biến. Điều này đảm bảo rằng chất lượng của giải pháp tốt nhất trong quần thể sẽ không bao giờ giảm qua các thế hệ.

a. Chi tiết thuật toán:

Chu trình của một thuật toán di truyền điển hình bao gồm các bước sau:

1. **Khởi tạo (Initialization):** Tạo một quần thể ban đầu gồm N cá thể một cách ngẫu nhiên.
2. **Vòng lặp chính (Main Loop):** Lặp lại các bước sau cho đến khi thỏa mãn điều kiện dừng:
 - *Dánh giá (Evaluation):* Tính toán giá trị thích nghi cho mỗi cá thể trong quần thể hiện tại.
 - *Lựa chọn (Selection):* Lựa chọn các cặp cha mẹ từ quần thể hiện tại. Các cá thể có độ thích nghi cao hơn có xác suất được chọn cao hơn.
 - *Lai ghép (Crossover):* Với mỗi cặp cha mẹ, thực hiện lai ghép với một xác suất nhất định để tạo ra các cá thể con.
 - *Đột biến (Mutation):* Áp dụng toán tử đột biến lên các cá thể con mới được tạo ra với một xác suất nhỏ.
 - *Tạo thế hệ mới (New Generation):* Quần thể mới được hình thành từ các cá thể con vừa được tạo. (Nếu sử dụng thuỷ tinh hoa, một số cá thể tốt nhất từ thế hệ trước cũng được đưa vào thế hệ mới).
3. **Kết thúc (Termination):** Quá trình dừng lại khi đạt đến số thế hệ tối đa, hoặc khi tìm thấy một giải pháp đủ tốt, hoặc khi độ thích nghi của quần thể không còn cải thiện đáng kể. Trả về cá thể tốt nhất trong quần thể cuối cùng.

b. Ví dụ về toán tử Lai ghép:

Toán tử lai ghép cho hai cha mẹ (giả sử có điểm lai ghép sau bit thứ 3) được minh họa như sau:

- Parent 1: 327 | 52411
- Parent 2: 247 | 48552

Sau khi lai ghép, hai con được tạo ra là:

- Child 1: 327 | 48552
- Child 2: 247 | 52411

Toán tử này cho phép các đặc tính tốt từ các phần khác nhau của không gian tìm kiếm được kết hợp lại với nhau.

Các tham số và ý nghĩa

Tham số	Ký hiệu	Giá trị	Ý nghĩa
Kích thước quần thể	N	50 - 200	Số lượng cá thể trong một thế hệ. Quần thể lớn hơn giúp tăng độ đa dạng nhưng làm chậm tốc độ tính toán.
Tỷ lệ lai ghép	p_c	0.6 - 0.9	Xác suất hai cá thể cha mẹ đã được chọn sẽ thực hiện lai ghép. Nếu không, chúng được sao chép trực tiếp sang thế hệ mới.
Tỷ lệ đột biến	p_m	0.001 - 0.05	Xác suất mỗi bit trong một nhiễm sắc thể sẽ bị thay đổi. Tỷ lệ thấp giúp duy trì các giải pháp tốt, tỷ lệ cao hơn giúp tăng khả năng khám phá.
Số thế hệ tối đa	MaxGenerations	100 - 1000	Điều kiện dừng của thuật toán, quy định số vòng lặp tối đa.

1.2.3 Simulated Annealing (SA)

Nguyên lý

Giải thuật xuất phát từ khái niệm *annealing* trong vật lý, mô tả quá trình nung nóng và làm nguội chậm của kim loại hoặc tinh thể nhằm tăng kích thước hạt và giảm khuyết điểm, từ đó giúp tinh thể đạt được trạng thái có năng lượng thấp nhất. Do đó, SA còn được gọi là *giải thuật mô phỏng luyện kim*.

Về bản chất, SA là một thuật toán tìm kiếm ngẫu nhiên (stochastic search) được sử dụng để tìm giá trị tối ưu toàn cục của một hàm mục tiêu. Thuật toán cho phép chấp nhận cả những nghiệm kém hơn với một xác suất nhất định để có thể thoát khỏi các cực trị địa phương, dựa trên nguyên lý Metropolis.

Chi tiết thuật toán

- **Bước 1:** Sinh ra một nghiệm ban đầu ngẫu nhiên x .
- **Bước 2:** Tính giá trị hàm mục tiêu $f(x)$.
- **Bước 3:** Sinh một nghiệm lân cận x' và tính $f(x')$.
- **Bước 4:** So sánh hai nghiệm:
 - Nếu $f(x') < f(x)$ thì chấp nhận nghiệm mới x' .
 - Nếu $f(x') > f(x)$ thì vẫn có thể chấp nhận x' với xác suất:

$$P = e^{-\Delta/T}$$

trong đó $\Delta = f(x') - f(x)$ và T là nhiệt độ hiện tại.

- **Bước 5:** Giảm nhiệt độ theo hàm giảm nhiệt.
- **Bước 6:** Kiểm tra điều kiện dừng. Nếu thỏa, kết thúc thuật toán; ngược lại, quay lại bước 3.

Quy trình khởi tạo nghiệm ban đầu

Ban đầu, thuật toán khởi tạo một nghiệm ngẫu nhiên trong không gian tìm kiếm thông qua hàm `initialize_solution()`. Nghiệm được sinh ngẫu nhiên trong phạm vi giới hạn của từng chiều (min_x, max_x) bằng phân phối đều (Uniform). Nghiệm này được xem như trạng thái ban đầu của hệ thống, tương tự trạng thái có năng lượng cao trong quá trình luyện kim. Giá trị hàm mục tiêu của nghiệm khởi tạo sẽ được lưu làm nghiệm tốt nhất hiện tại (`best_solution`).

Chọn nhiệt độ ban đầu

Nhiệt độ ban đầu T_0 được khởi tạo bằng một giá trị cố định (ví dụ 100 trong chương trình). Nhiệt độ này điều khiển xác suất chấp nhận các nghiệm có giá trị kém hơn trong giai đoạn đầu. Khi T_0 càng cao, thuật toán càng dễ chấp nhận nghiệm xấu hơn, giúp khám phá không gian tìm kiếm rộng và tránh mắc kẹt tại cực trị cục bộ.

Chiến lược sinh nghiệm lân cận

Ở mỗi vòng lặp, một nghiệm lân cận x' được sinh ra từ nghiệm hiện tại x bằng cách thêm nhiễu ngẫu nhiên có phân phối chuẩn:

$$x' = x + \mathcal{N}(0, 1) \times T \times \text{neighbor_scale}$$

Nhiệt độ T và hệ số `neighbor_scale` điều chỉnh độ lớn của bước nhảy. Sau khi sinh nghiệm, giá trị từng biến được giới hạn lại (`np.clip`) để đảm bảo nằm trong phạm vi hợp lệ.

Hàm giảm nhiệt

Nhiệt độ T được sử dụng để điều khiển xác suất chấp nhận các nghiệm kém hơn. Khi nhiệt độ cao, thuật toán dễ chấp nhận các nghiệm xấu hơn, giúp khám phá không gian tìm kiếm rộng hơn. Khi nhiệt độ giảm dần, thuật toán dần tập trung vào vùng nghiệm tốt và hội tụ về nghiệm tối ưu. Hàm giảm nhiệt thường có dạng:

$$T_{k+1} = \alpha \times T_k \quad \text{với } 0 < \alpha < 1$$

trong đó α là hệ số làm nguội (cooling rate).

Hàm chấp nhận (Acceptance Probability)

Sau khi tính được giá trị hàm mục tiêu của nghiệm mới $f(x')$, giải thuật quyết định có chấp nhận hay không dựa theo tiêu chuẩn Metropolis:

$$P = \begin{cases} 1, & \text{nếu } \Delta < 0 \\ e^{-\Delta/T}, & \text{nếu } \Delta \geq 0 \end{cases}$$

trong đó $\Delta = f(x') - f(x)$ là chênh lệch năng lượng (hay giá trị hàm mục tiêu). Nếu $\Delta < 0$, nghiệm mới tốt hơn sẽ được chấp nhận ngay. Ngược lại, nghiệm xấu hơn vẫn có thể được chấp nhận ngẫu nhiên với xác suất $e^{-\Delta/T}$, giúp thuật toán thoát khỏi cực trị địa phương.

Cooling Schedule

Sau mỗi vòng lặp, nhiệt độ được giảm dần theo quy luật hình học:

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

trong đó $0 < \alpha < 1$ là hệ số làm nguội (`cooling_rate`). Giá trị α càng nhỏ, nhiệt độ giảm càng nhanh, dẫn đến hội tụ sớm nhưng dễ mắc kẹt tại cực trị địa phương. Ngược lại, α gần 1 giúp thuật toán khám phá lâu hơn nhưng mất thời gian hội tụ. Trong chương trình, α thường chọn trong khoảng $[0.9, 0.99]$ để cân bằng giữa khám phá và khai thác.

2 TRIỂN KHAI VÀ THIẾT KẾ

2.1 Cấu trúc chương trình và môi trường

Môi trường và các thư viện

Các thuật toán tối ưu được cài đặt và thực nghiệm trong môi trường Python phiên bản 3.8 trở lên. Các thư viện chính được sử dụng bao gồm:

- **NumPy** ($>= 1.21.0$): hỗ trợ tính toán ma trận và các phép toán khoa học cơ bản.
- **Pandas** ($>= 1.3.0$): xử lý và phân tích dữ liệu dạng bảng (DataFrame).
- **Matplotlib** ($>= 3.4.0$): trực quan hóa dữ liệu và vẽ đồ thị.
- **Seaborn** ($>= 0.11.0$): trực quan hóa thống kê với giao diện mở rộng cho Matplotlib.
- **PyYAML** ($>= 6.0$): dùng để đọc và xử lý các tệp cấu hình định dạng YAML.

Cấu trúc thư mục

2.2 Thiết lập tham số

ACO

Tham số mặc định

- **Hệ số ảnh hưởng α và β :** Giá trị phổ biến thường được chọn là $\alpha = 1$ và β trong khoảng $[2, 5]$. Stützle và Hoos (2000) thường đề xuất $\alpha = 1, \beta = 2$ cho MMAS khi áp dụng cho bài toán TSP. [1]
- **Tốc độ bay hơi ρ :** Giá trị ρ nằm trong khoảng $(0, 1]$. Trong MMAS, giá trị này thường được chọn tương đối nhỏ để thúc đẩy việc khai thác kỹ hơn, ví dụ $\rho = 0.1$.
- **Số lượng kiến m :** Thường được chọn bằng với số lượng đỉnh (n) trong đồ thị của bài toán ($m = n$). (*Trong bài làm của nhóm, đối với bài toán TSP, khi tinh chỉnh tham số qua file config, cho số lượng kiến bằng -1 sẽ tự động chỉnh số lượng kiến bằng với số thành phố của từng bài toán.*)
- **Tần suất cập nhật sử dụng tốt nhất toàn cục u_{gb} :** Thường bằng 1, khi đó, luôn luôn sử dụng lời giải tốt nhất trên toàn cục để cập nhật pheromones. Khi lớn hơn 1, thì cứ sau mỗi u_{gb} vòng lặp, sử dụng lời giải tốt nhất trên toàn cục để cập nhật pheromones thay vì sử dụng lời giải tốt nhất trong vòng lặp đó.
- **Nguồng bế tắc stagnation** Sau *stagnation* vòng lặp, nếu không có thay đổi nào trên kết quả tốt nhất thì thực hiện cài đặt lại pheromones trên toàn bộ đường đi. Trong bài làm, được mặc định là 50.

PSO

Tham số mặc định

- **Inertia weight w** : được chọn trong khoảng [0.9, 1.2] theo Shi và Eberhart (1998). Inertia weight giúp kiểm soát cân bằng giữa *global search* và *local search*: giá trị lớn có ý nghĩa khi tìm kiếm toàn cục, giá trị nhỏ có ý nghĩa khi tìm kiếm cục bộ. Nhiều nghiên cứu cũng đề xuất các biến thể với w giảm tuyến tính theo số vòng lặp, hoặc điều chỉnh dựa trên trạng thái quần thể, tốc độ trung bình, độ đa dạng, v.v.
- **Learning factors c_1 và c_2** : đại diện cho trọng số của các thành phần tăng tốc ngẫu nhiên kéo mỗi particle về *pBest* và *gBest*. Giá trị phổ biến được chọn là $c_1 = c_2 = 1.49445$ để đảm bảo hội tụ của PSO (theo Clerc và Kennedy 2002). Nhiều nghiên cứu khác cũng đã thử nghiệm các giá trị khác hoặc các hệ số thay đổi theo thời gian, nhưng giá trị này được đánh giá là cân bằng giữa tìm kiếm cục bộ và toàn cục.

Tham số phân tích sensitivity

- **Inertia weight $w = [0.4, 0.6, 0.8, 0.9, 0.95]$** : Các giá trị này được lựa chọn dựa trên đề xuất của Eberhart và Shi (2001), trong đó trọng số quan tính được xác định ngẫu nhiên theo công thức:

$$w \in \left[0.5 + \frac{\text{rnd}}{2.0} \right]$$

Do đó, việc cài đặt các giá trị w khác nhau giúp xem xét và đánh giá ảnh hưởng của hệ số này đến hiệu năng của PSO, đồng thời xác định giá trị w tối ưu cho thuật toán.

- **Learning factors $c_1 = c_2 = [0.5, 1.0, 1.49445, 2.0, 2.5]$** : Trong nhiều nghiên cứu, các hệ số c_1 và c_2 thường được chọn là 2.0, tuy nhiên giá trị 1.49445 được Clerc và Kennedy (2002) chứng minh đảm bảo tính hội tụ của PSO. Vì vậy, việc khảo sát nhiều giá trị khác nhau của c_1 và c_2 giúp đánh giá ảnh hưởng của chúng đến khả năng tìm kiếm và tốc độ hội tụ của thuật toán.

Nguồn cấu hình: Tất cả tham số được định nghĩa trong `configs/algorithms/`, áp dụng **đồng nhất** cho tất cả các bài toán (Ackley, Sphere, Rastrigin).

ABC

File cấu hình: `configs/algorithms/abc.yaml`

Bảng 2.1: Thiết lập tham số cho Artificial Bee Colony (ABC).

Tham số	Giá trị	Ý nghĩa
colony_size	50	Tổng số ong trong đàn (bao gồm employed bees, onlooker bees, và scout bees)
max_iterations	100	Số vòng lặp tối đa của thuật toán
limit	20	Nguồn từ bỏ: số lần thử cải thiện một nguồn thức ăn trước khi bị abandoned bởi scout bees

Cơ chế hoạt động:

- **Employed Bees Phase:** 50 ong được phân công cho 50 nguồn thức ăn (food sources), mỗi ong thực hiện local search quanh nguồn của mình.
- **Onlooker Bees Phase:** Các onlooker bees chọn nguồn thức ăn dựa trên roulette wheel selection theo fitness, sau đó khai thác thêm.
- **Scout Bees Phase:** Nếu một nguồn thức ăn không cải thiện sau 20 lần thử (limit), nó bị từ bỏ và được thay thế bằng một nguồn ngẫu nhiên mới.

Cân bằng Exploration-Exploitation:

- *Exploitation:* Employed và Onlooker bees khai thác các vùng hứa hẹn
- *Exploration:* Scout bees (kích hoạt khi limit đạt ngưỡng) khám phá không gian mới

CS

Cấu hình chung:

Listing 2.1: Cấu hình cơ bản Cuckoo Search

```

1 algorithm:
2   name: "Cuckoo Search"
3   n_nests: 25
4   pa: 0.25
5   beta: 1.5
6   alpha: 0.01
7   max_iter: 300

```

Cấu hình theo từng bài toán:

1. Sphere Function (unimodal):

- $n_{nests} = 25$: Đủ để khám phá không gian đơn giản
- $p_a = 0.25$: Giá trị cân bằng, không cần exploration mạnh
- $\alpha = 0.01$: Nhỏ để exploitation tốt, hội tụ nhanh
- $max_iter = 300$: Đủ để hội tụ hoàn toàn

2. Ackley Function (multimodal):

- $n_{nests} = 50$: Tăng gấp đôi để khám phá nhiều vùng
- $p_a = 0.3$: Cao hơn để tránh kẹt ở cực trị địa phương
- $\alpha = 0.1$: Lớn hơn 10x để bước nhảy xa, khám phá hiệu quả
- $max_iter = 500$: Tăng vì Ackley khó hội tụ hơn

Nguyên tắc chung:

- Hàm đơn modal $\rightarrow p_a$ thấp, α nhỏ, ít iterations
- Hàm đa modal $\rightarrow p_a$ cao, α lớn, nhiều iterations
- $\beta = 1.5$ luôn giữ nguyên (giá trị tối ưu lý thuyết)

2.2.1 Các bài toán

Rời rạc: TSP

Mục tiêu: Cài đặt lớp mô phỏng bài toán *Travelling Salesman Problem (TSP)*, cho phép đọc dữ liệu từ tệp CSV, tính toán ma trận khoảng cách giữa các thành phố, và ước lượng độ dài hành trình bằng thuật toán *Nearest Neighbor*.

Nội dung chính:

- **Khởi tạo (`__init__`):** Đọc dữ liệu tọa độ từ tệp CSV, tạo ma trận khoảng cách và ma trận heuristic $\eta = \frac{1}{d_{ij}}$.
- **Hàm `_load_from_csv()`:** Đọc tọa độ (x, y) từ tệp CSV, lưu vào `self.coordinates` và xác định số lượng thành phố (`self.dim`).
- **Hàm `_compute_distance_matrix()`:** Tính khoảng cách Euclid giữa mọi cặp điểm để tạo ma trận đối xứng.
- **Hàm `_nearest_neighbor_tour_length()`:** Mô phỏng hành trình bắt đầu từ một thành phố ngẫu nhiên, chọn liên tiếp thành phố gần nhất chưa thăm, và trả về tổng quãng đường của chu trình.

Kết luận: Lớp `TSPProblem` giúp xử lý linh hoạt dữ liệu TSP, hỗ trợ tính toán khoảng cách và đánh giá nhanh chất lượng lời giải bằng phương pháp heuristic đơn giản.

Liên tục: Sphere, Ackley và Rastrigin

1. Sphere function

- **Hàm số:**

$$f(\mathbf{x}) = \sum_{i=1}^n x_i^2 \quad (2.1)$$

- **Miền giá trị:**

$$x_i \in [-100, 100], \quad i = 1, 2, \dots, n$$

Khoảng giá trị này thường được sử dụng trong hầu hết các nghiên cứu chuẩn hoá để đảm bảo không gian tìm kiếm đủ lớn cho các thuật toán tối ưu toàn cục, đồng thời vẫn đảm bảo tính ổn định khi tính toán bằng số thực.

- **Global minimum:**

$$f(\mathbf{x}^*) = 0 \quad \text{khi } \mathbf{x}^* = (0, 0, \dots, 0)$$

- **Số chiều thử nghiệm:** $n = 10$

Giá trị $n = 10$ được chọn vì đây là cấu hình tiêu chuẩn trong nhiều thí nghiệm metaheuristic (chẳng hạn như PSO, GA, SA), giúp đánh giá khả năng hội tụ mà không làm tăng chi phí tính toán quá cao. Khi n lớn hơn (ví dụ 30, 50 hoặc 100), độ phức tạp sẽ tăng mạnh, nhưng xu hướng hội tụ nhìn chung vẫn tương tự.

- **Ý nghĩa:** Hàm Sphere là hàm lồi, trơn và chỉ có một cực tiểu toàn cục. Nó được sử dụng để đánh giá khả năng hội tụ cơ bản của thuật toán tối ưu mà không bị ảnh hưởng bởi cực trị cục bộ.

2. Ackley function

- **Hàm số:**

$$f(\mathbf{x}) = -20 \exp \left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \right) - \exp \left(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i) \right) + 20 + e \quad (2.2)$$

- **Miền giá trị:**

$$x_i \in [-32.768, 32.768], \quad i = 1, 2, \dots, n$$

Đây là miền được thiết lập theo định nghĩa chuẩn của Ackley trong bộ kiểm thử chuẩn của *CEC Benchmark Functions*. Khoảng này đủ rộng để hàm thể hiện rõ đặc tính nhiều cực trị cục bộ xen kẽ với cực trị toàn cục.

- **Global minimum:**

$$f(\mathbf{x}^*) = 0 \quad \text{khi } \mathbf{x}^* = (0, 0, \dots, 0)$$

- **Số chiều thử nghiệm:** $n = 10$

Tương tự như Sphere, việc chọn $n = 10$ giúp đánh giá khả năng **thoát khỏi cực trị cục bộ** trong không gian đa chiều nhưng vẫn đảm bảo thời gian huấn luyện hợp lý.

- **Ý nghĩa:** Hàm Ackley có nhiều cực trị cục bộ, giúp kiểm tra khả năng cân bằng giữa khai phá và khai thác (exploration-exploitation) của thuật toán. Đây là bài toán phi tuyến điển hình dùng để kiểm tra hiệu quả tối ưu hóa toàn cục.

3. Rastrigin function

- **Hàm số:**

$$f(\mathbf{x}) = 10n + \sum_{i=1}^n [x_i^2 - 10 \cos(2\pi x_i)] \quad (2.3)$$

- **Miền giá trị:**

$$x_i \in [-5.12, 5.12], \quad i = 1, 2, \dots, n$$

Khoảng này được sử dụng theo định nghĩa gốc của Rastrigin function, đảm bảo bề mặt hàm có mật độ cực trị cục bộ dày đặc nhưng vẫn đủ nhỏ để các thuật toán có thể tìm ra nghiệm tối ưu trong số vòng lặp giới hạn.

- **Global minimum:**

$$f(\mathbf{x}^*) = 0 \quad \text{khi} \quad \mathbf{x}^* = (0, 0, \dots, 0)$$

- **Số chiều thử nghiệm:** $n = 10$

Với $n = 10$, hàm Rastrigin đủ phức tạp để thử thách khả năng tìm kiếm của PSO, GA hoặc SA, nhưng vẫn có thể trực quan hóa và phân tích được.

- **Ý nghĩa:** Hàm Rastrigin có nhiều cực trị cục bộ phân bố đều, đại diện cho những bài toán có nhiều nghiệm khả dĩ. Nó dùng để kiểm tra mức độ **khai thác không gian tìm kiếm hiệu quả** của thuật toán.

Điều kiện dừng

Điều kiện dừng quy định khi nào thuật toán tối ưu hóa kết thúc quá trình tìm kiếm. Trong dự án này, các thuật toán sử dụng điều kiện dừng chính là **số vòng lặp tối đa cố định**, kết hợp với một số cơ chế phụ.

Bảng 2.2: So sánh điều kiện dừng của 7 thuật toán

Thuật toán	Điều kiện chính	Số lần lặp	Loại bài toán	Cơ chế phụ
PSO	Fixed iterations	1000	Continuous	Không
ABC	Fixed iterations	100	Continuous	Scout bees (limit=20)
FA	Fixed generations	100	Continuous	Alpha cooling (0.95)
CS	Fixed iterations	100	Continuous	Discovery phase (pa=0.25)
GA	Fixed generations	100	Continuous	Elitism (2 best)
SA	Fixed iterations	100	Continuous	Temperature cooling (0.95)
ACO	Fixed iterations	100	Discrete (TSP)	Stagnation (50)

Nhận xét:

1. Tất cả thuật toán sử dụng Fixed Budget:

- Điều kiện dừng chính là số vòng lặp/generations cố định
- Không có early stopping based on convergence hoặc fitness threshold
- Ưu điểm: Công bằng khi so sánh (cùng computational budget)
- Nhược điểm: Có thể lãng phí computation nếu đã hội tụ sớm

2. PSO có nhiều iterations nhất (1000 vs 100):

- Lý do: PSO update nhanh hơn (chỉ update velocity & position)
- Các thuật toán khác phức tạp hơn (selection, crossover, attraction, ...)
- Để công bằng về computational time, cần xem xét cost per iteration

3. Cơ chế phụ không phải điều kiện dừng:

- ABC limit, FA alpha cooling, CS discovery, SA temperature
- Đây là các *adaptive mechanisms* hoặc *diversification strategies*

- Chúng ảnh hưởng đến **cách tìm kiếm**, không phải **khi nào dừng**

4. Không có convergence-based stopping:

- Không dừng khi fitness change $< \epsilon$ trong N iterations
- Không dừng khi đạt fitness target (e.g., $< 10^{-6}$)
- Không dừng khi diversity quá thấp (swarm collapsed)
- Lý do: Design choice để đảm bảo fair comparison

3 PHƯƠNG PHÁP THỰC NGHIỆM

3.1 Phương pháp đánh giá

Đánh giá và so sánh hiệu quả của các thuật toán swarm với các thuật toán classical bằng các phương pháp sau:

Convergence Speed

Định nghĩa: Convergence speed thể hiện tốc độ mà thuật toán tiến đến nghiệm tối ưu (global minimum hoặc local minimum) theo số vòng lặp hoặc thời gian.

Cách đo: Đường hội tụ được vẽ bằng giá trị trung bình của hàm mục tiêu tốt nhất theo từng vòng lặp:

$$f_{\text{best}}(t) = \min_{1 \leq i \leq N} f_i(t)$$

với N là số cá thể (hoặc nghiệm) trong quần thể tại vòng lặp t .

Ý nghĩa: Thuật toán có *convergence speed* nhanh chứng tỏ khả năng khai thác (exploitation) hiệu quả, nhưng nếu hội tụ quá sớm có thể sẽ dẫn đến *local optimum*.

Mean Time (s)

Định nghĩa: Là thời gian trung bình (tính bằng giây) để thuật toán hoàn thành một lần tối ưu trên một hàm chuẩn.

Cách tính:

$$\text{Mean Time} = \frac{1}{R} \sum_{r=1}^R t_r$$

với t_r là thời gian thực thi ở lần chạy thứ r và R là tổng số lần chạy (thường $R = 30$).

Ý nghĩa: Thể hiện chi phí tính toán và hiệu quả thực thi của thuật toán. Thuật toán có *mean time* thấp thường có hiệu suất cao hơn trong các ứng dụng thời gian thực.

Mean Fitness

Định nghĩa: Là giá trị trung bình của hàm mục tiêu (fitness function) thu được sau nhiều lần chạy độc lập.

Công thức:

$$\text{Mean Fitness} = \frac{1}{R} \sum_{r=1}^R f_r$$

với f_r là giá trị tốt nhất đạt được trong lần chạy thứ r .

Ý nghĩa: Thể hiện mức độ tối ưu trung bình của thuật toán. Giá trị càng thấp (đối với bài toán minimization) thì chất lượng nghiệm càng cao.

Std Fitness

Định nghĩa: Là độ lệch chuẩn của các giá trị fitness qua nhiều lần chạy.

Công thức:

$$\text{Std Fitness} = \sqrt{\frac{1}{R-1} \sum_{r=1}^R (f_r - \bar{f})^2}$$

với \bar{f} là giá trị trung bình của fitness.

Ý nghĩa: Thể hiện độ ổn định của thuật toán. Giá trị *Std Fitness* nhỏ cho thấy thuật toán có khả năng tái lập kết quả tốt, ít bị phụ thuộc vào yếu tố ngẫu nhiên.

Robustness

Định nghĩa: Là khả năng của thuật toán duy trì hiệu suất ổn định trên các hàm mục tiêu khác nhau hoặc dưới các điều kiện khởi tạo ngẫu nhiên khác nhau.

Cách đánh giá:

$$\text{Robustness} = \frac{\text{Số lần đạt global optimum}}{\text{Tổng số lần chạy}} \times 100\%$$

hoặc có thể dựa trên sự ổn định của *Std Fitness* và *Mean Fitness*.

Ý nghĩa: Thuật toán càng *robust* khi kết quả không bị ảnh hưởng đáng kể bởi các thay đổi trong tham số hoặc dữ liệu đầu vào.

Scalability

Định nghĩa: Scalability (khả năng mở rộng) mô tả mức độ mà hiệu suất của thuật toán bị ảnh hưởng khi tăng kích thước bài toán (ví dụ: số chiều n hoặc số cá thể).

Cách đo: Thực hiện thí nghiệm với nhiều giá trị n khác nhau (ví dụ $n = 10, 30, 50$) và so sánh xu hướng của thời gian, mean fitness và tốc độ hội tụ.

Ý nghĩa: Thuật toán có *scalability* tốt sẽ duy trì hoặc chỉ giảm hiệu suất nhẹ khi kích thước bài toán tăng, thể hiện khả năng áp dụng cho các vấn đề lớn thực tế.

3.2 Visualize

3.2.1 Convergence

Mục đích

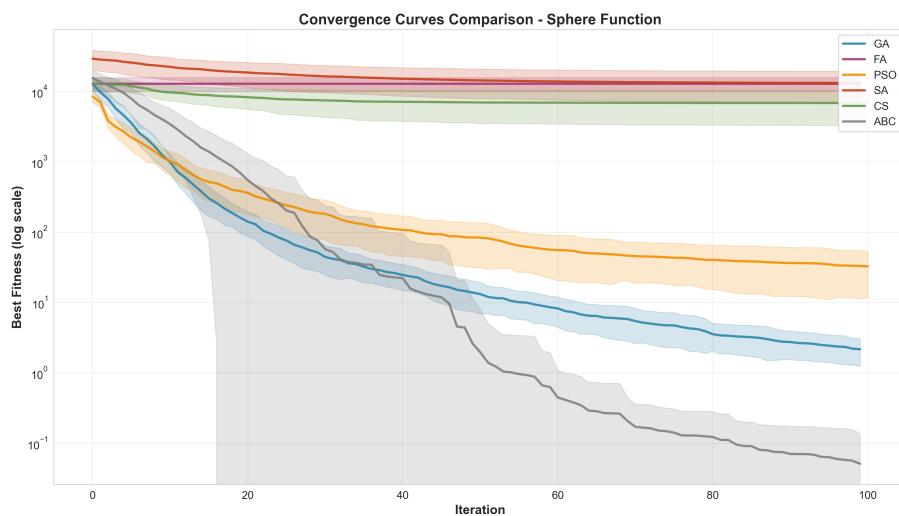
- **Dánh giá tốc độ hội tụ:** Thuật toán nào đạt được nghiệm tốt nhanh nhất?
- **Phát hiện premature convergence:** Thuật toán có bị kẹt ở cực trị địa phương không?
- **So sánh hiệu suất:** Thuật toán nào đạt fitness tốt nhất và ổn định nhất?
- **Hiểu hành vi thuật toán:** Phân biệt giai đoạn exploration và exploitation

Các loại biểu đồ

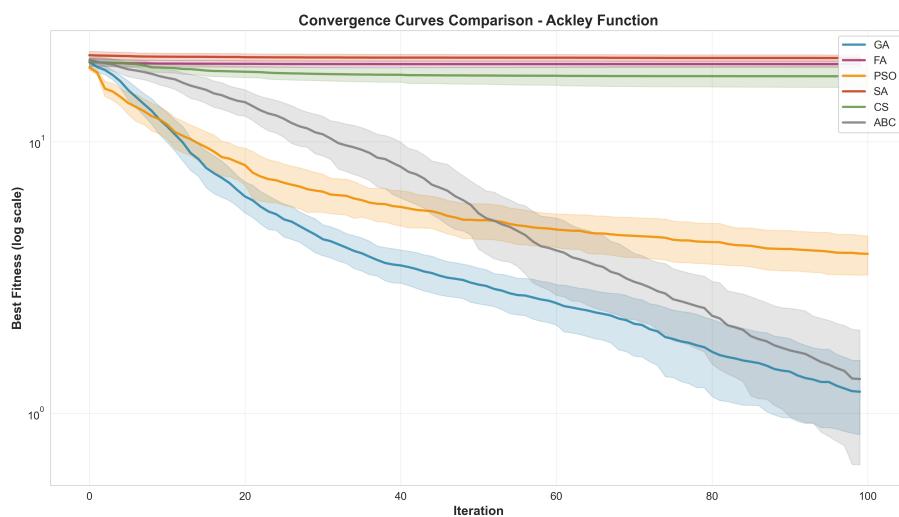
1. Convergence Plot cơ bản - Best Fitness vs Iterations:

Đây là dạng biểu đồ phổ biến nhất để theo dõi quá trình hội tụ của thuật toán.
Thành phần:

- Trục X: Số iterations từ 0 đến max_iter (thường 300-500)
- Trục Y: Best fitness value (log scale nếu giá trị chênh lệch lớn)
- Mỗi thuật toán: Một đường màu khác nhau
- Thể hiện giá trị fitness TỐT NHẤT tìm được tại mỗi iteration



Hình 3.1: So sánh hội tụ của các thuật toán trên Sphere Function.



Hình 3.2: So sánh hội tụ của các thuật toán trên Ackley Function.

3.2.2 Comparative Performance

Tổng quan

Mục đích:

Phân tích so sánh hiệu suất nhằm đánh giá:

1. **Chất lượng nghiệm/ lời giải** (Solution Quality) qua Best/Mean Fitness
2. **Tốc độ hội tụ** (Convergence Speed) - số iteration cần để đạt ngưỡng
3. **Hiệu quả tính toán** (Time Efficiency) - thời gian thực thi
4. **Độ ổn định** (Robustness) - tính nhất quán qua nhiều lần chạy
5. **Khả năng mở rộng** (Scalability) - hiệu suất khi tăng số chiều

Việc so sánh này giúp xác định thuật toán phù hợp cho từng loại bài toán và điều kiện thực tế.

Phương pháp So sánh

1. Thiết kế Thử nghiệm

- **Benchmark Functions:** Sphere (unimodal), Ackley (multimodal), Rastrigin (highly multimodal)
- **Số chiều:** 10D (standard), scalability test: 5D, 10D, 20D, 30D
- **Số lần chạy:** 30 independent runs per algorithm per problem
- **Budget:** 100 iterations × population size
- **Random seeds:** Khác nhau cho mỗi run để đảm bảo tính độc lập

2. Performance Metrics

Bảng 3.1: Định nghĩa các chỉ số hiệu suất

Metric	Công thức	Ý nghĩa
Best Fitness	$\min_{i=1}^N f(x_i^{best})$	Giải pháp tốt nhất tìm được
Mean Fitness	$\frac{1}{N} \sum_{i=1}^N f(x_i^{best})$	Hiệu suất trung bình
Std Fitness	$\sqrt{\frac{1}{N} \sum_{i=1}^N (f(x_i^{best}) - \mu)^2}$	Độ biến động
Convergence Speed	t_{conv} where $f(x_t) \leq \epsilon$	Số iteration để hội tụ
Robustness	$1 - \frac{\text{Std}}{\text{Mean}}$ (normalized)	Tính ổn định (0-1)
Scalability Coeff	Slope of $T(d)$ vs d	Tốc độ tăng thời gian

3. Paradigm Classification

Các thuật toán được phân loại thành hai nhóm chính:

- **Swarm-based:** PSO, FA, CS, ABC
- **Classical:** GA, SA

Cấu trúc Dữ liệu và Visualizations

1. Data Files

Bảng 3.2: Cấu trúc files kết quả performance

File	Nội dung	Mục đích
*_metrics.csv	Best, Mean, Std, Conv Speed, Time, Robustness	So sánh tổng hợp
*_scalability.csv	Scalability Complexity	Coefficient, Time Phân tích scalability

2. Visualization Categories

Ba nhóm visualization chính được tạo ra:

1. Overview (visualizations/continuous/performance/overview/)

- *_radar.png: Performance profile cho từng problem
- performance_comparison.png: So sánh 4 metrics across problems
- scalability_comparison.png: Time vs dimensions
- statistical_summary.png: Heatmap normalized scores

2. Swarm vs Classical (.../swarm_classic_comparison/)

- side_by_side_comparison.png: Direct comparison
- category_averages.png: Paradigm averages
- paradigm_strengths_heatmap.png: Normalized strengths
- paradigm_performance_profiles.png: Radar charts
- scalability_paradigm_comparison.png: Scalability by paradigm

3. Swarm Only (.../swarm_only/)

- convergence_speed_*.png: Convergence analysis
- computational_time_*.png: Time analysis
- robustness_*.png: Robustness comparison
- scalability_*.png: Scalability within swarm

Các Dạng Visualization

1. Radar Chart (Performance Profile)

Đặc điểm:

- **Dạng:** Polar plot với 4 trục (Quality, Speed, Time Efficiency, Robustness)
- **Mỗi đường:** Một thuật toán, kết nối 4 điểm metrics
- **Vùng tô màu:** Diện tích càng lớn = performance càng tốt

- **Normalized scores:** [0, 1], cao hơn = tốt hơn

Cách đọc:

- *Dường tròn đều:* Thuật toán cân bằng trên tất cả metrics
- *Hình sao nhọn:* Mạnh ở một số metrics, yếu ở metrics khác
- *Diện tích lớn nhất:* Thuật toán tốt nhất overall

2. Bar Chart Comparison

Đặc điểm:

- **Layout:** 2×2 subplots (Best Fitness, Conv Speed, Time, Robustness)
- **Grouped bars:** Mỗi algorithm có 3 bars (Sphere, Ackley, Rastrigin)
- **Scales:** Log scale cho Best Fitness, linear cho các metrics khác
- **Error bars:** Hiển thị trong Time plot (std deviation)

Cách đọc:

- *Best Fitness:* Càng thấp càng tốt (logarithmic scale)
- *Convergence Speed:* Càng thấp càng nhanh
- *Time:* Càng thấp càng hiệu quả
- *Robustness:* Càng cao càng ổn định (max = 1.0)

3. Heatmap (Paradigm Strengths)

Đặc điểm:

- **Dạng:** Heatmap 2×4 (2 paradigms \times 4 metrics)
- **Màu sắc:** RdYlGn colormap (Red-Yellow-Green)
- **Giá trị:** Normalized scores [0, 1]
- **Annotations:** Giá trị số hiển thị trong mỗi ô

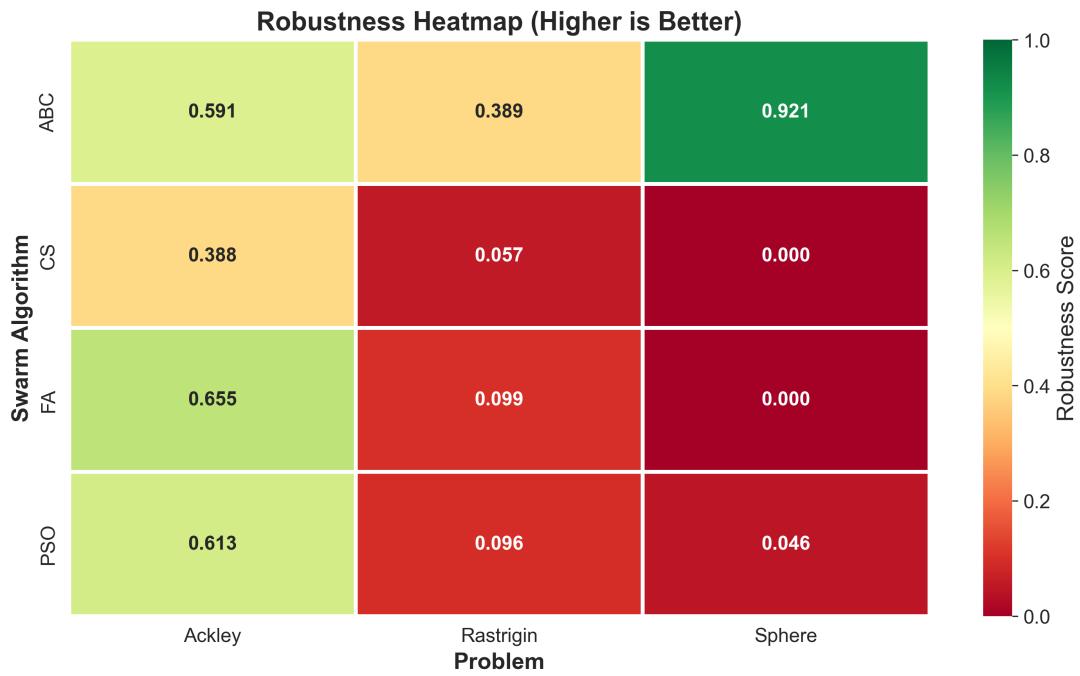
Cách đọc:

- **Đỏ (0.0-0.4):** Hiệu suất kém
- **Vàng (0.4-0.6):** Hiệu suất trung bình
- **Xanh (0.6-1.0):** Hiệu suất tốt

4. Convergence Speed Biểu đồ thể hiện số vòng lặp cần thiết để thuật toán hội tụ đến nghiệm tốt nhất. Giá trị càng thấp thể hiện tốc độ hội tụ càng nhanh.

5. Computational Time Biểu đồ so sánh thời gian thực thi trung bình của mỗi thuật toán trên các hàm benchmark.

6. Robustness Chỉ số đo mức độ ổn định giữa các lần chạy. Giá trị gần 1 thể hiện tính ổn định cao.



Hình 3.3: Biểu đồ heatmap thể hiện độ ổn định (Higher is Better)

7. Scalability Analysis

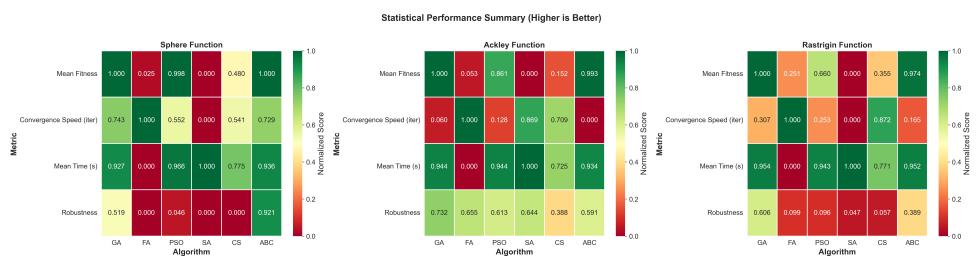
Đặc điểm:

- **Trục X:** Problem dimensions (5D, 10D, 20D, 30D)
- **Trục Y:** Execution time (seconds)
- **Mối đường:** Một thuật toán
- **Markers:** Điểm dữ liệu thực tế

Cách đọc:

- *Dường phẳng*: Scalability tốt (không tăng nhanh theo dimensions)
- *Dường dốc*: Scalability kém (tăng nhanh khi tăng dimensions)
- *Slope (hệ số góc)*: Scalability coefficient (càng thấp càng tốt)

8. Statistical Summary Heatmap



Hình 3.4: Statistical summary heatmap với normalized scores

Đặc điểm:

- **Rows:** Metrics (Mean Fitness, Conv Speed, Time, Robustness)
- **Columns:** Algorithms
- **Values:** Normalized [0, 1], higher = better
- **Purpose:** Quick visual comparison across all dimensions

3.2.3 3D Surface

Mục đích

Biểu đồ 3D surface plots (landscape visualization) cung cấp cái nhìn trực quan về:

1. **Trực quan hóa bề mặt hàm mục tiêu:** Hiểu được địa hình (topology) của không gian tìm kiếm
2. **Đánh giá độ khó của bài toán:** Bề mặt trơn → dễ tối ưu; Bề mặt gồ ghề → khó
3. **Minh họa hành vi thuật toán:** Xem thuật toán di chuyển như thế nào trên bề mặt

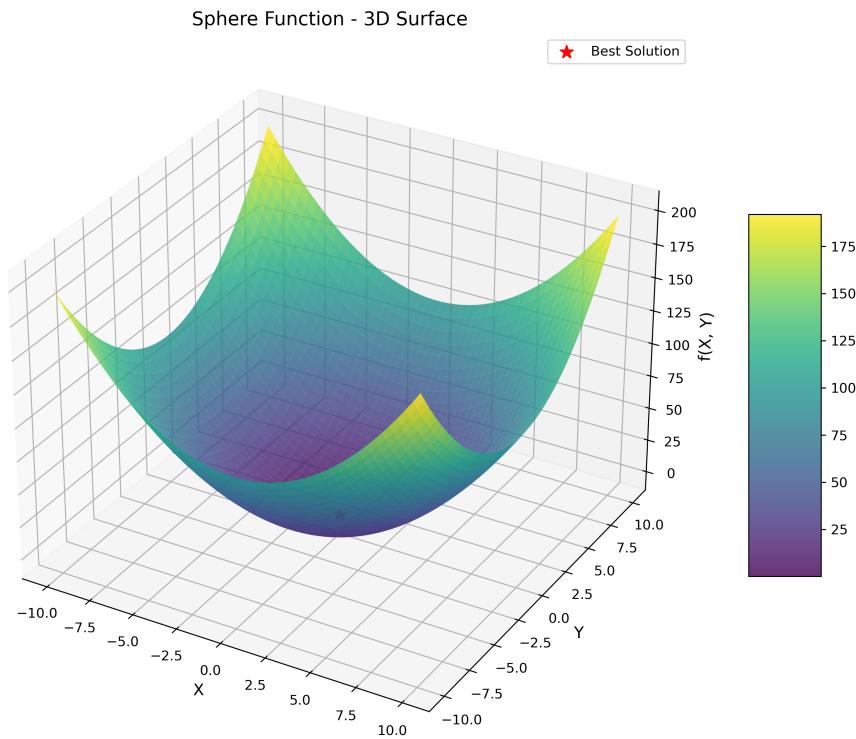
1. Sphere Function:

Công thức:

$$f(x) = \sum_{i=1}^D x_i^2 \quad (3.1)$$

Đặc điểm:

- **Loại:** Unimodal (đơn modal)
- **Cực tiểu toàn cục:** $f(0, 0, \dots, 0) = 0$
- **Bounds:** Thường $[-5, 5]$ hoặc $[-10, 10]$
- **Hình dạng 3D:** Bát parabol hoàn hảo (perfect paraboloid)
- **Gradient:** Tăng đều từ tâm ra ngoài, luôn hướng về tâm
- **Đường đồng mức:** Các vòng tròn đồng tâm



Hình 3.5: 3D Surface của Sphere Function.

Ý nghĩa: Sphere là hàm test ĐƠN GIẢN NHẤT. Không có local minima, thuật toán tốt phải đạt fitness xấp xỉ 0.

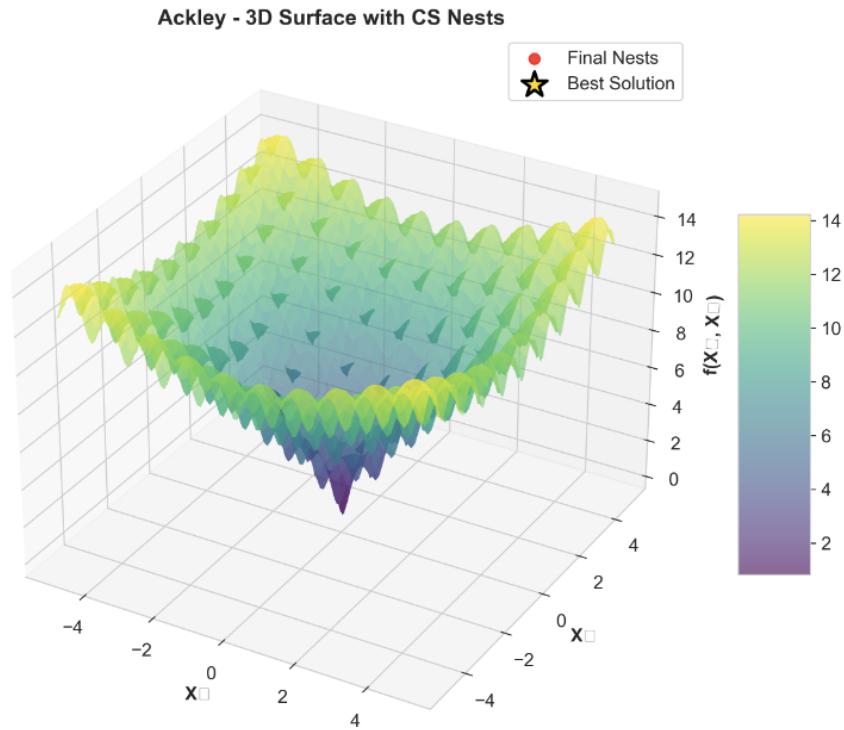
2. Ackley Function:

Công thức:

$$f(x) = -20 \exp \left(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^D x_i^2} \right) - \exp \left(\frac{1}{D} \sum_{i=1}^D \cos(2\pi x_i) \right) + 20 + e \quad (3.2)$$

Đặc điểm:

- **Loại:** Multimodal (đa modal)
- **Cực tiểu toàn cục:** $f(0, 0, \dots, 0) = 0$
- **Bounds:** Thường $[-5, 5]$ hoặc $[-32.768, 32.768]$
- **Hình dạng 3D:** Chảo lớn ở giữa với bề mặt gợn sóng (ripples)
- **Cấu trúc:** Có một hố sâu ở tâm (global minimum), xung quanh có vô số hố nhỏ (local minima)
- **Gradient:** Phức tạp, nhiều vùng flat (plateau)



Hình 3.6: 3D Surface của Ackley Function.

Ý nghĩa: Ackley test khả năng escape local minima. Fitness < 5 là acceptable, < 1 là xuất sắc.

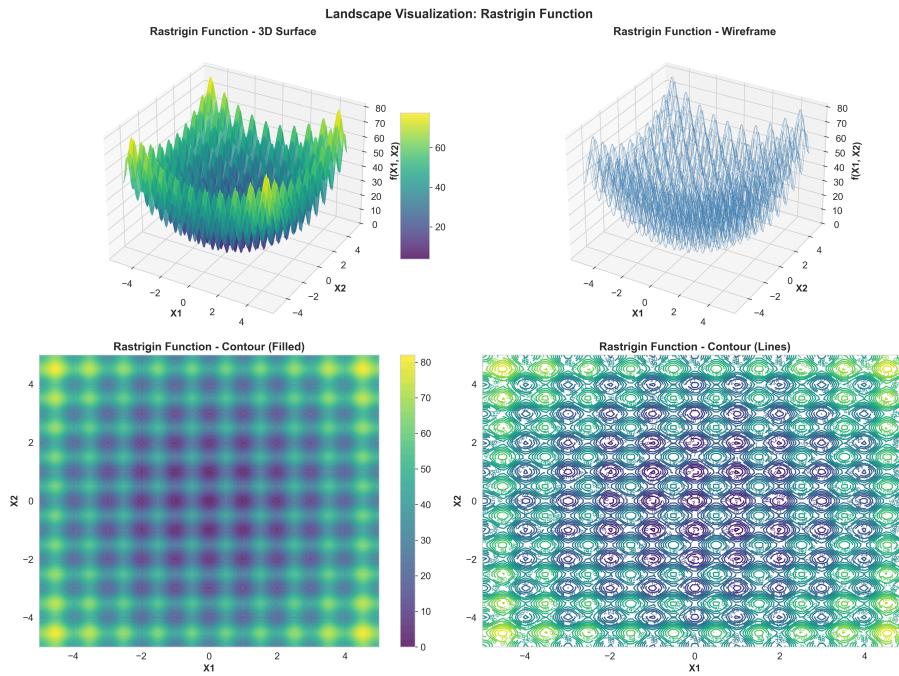
3. Rastrigin Function:

Công thức:

$$f(x) = 10D + \sum_{i=1}^D [x_i^2 - 10 \cos(2\pi x_i)] \quad (3.3)$$

Đặc điểm:

- **Loại:** Multimodal (đa cực trị)
- **Cực tiểu toàn cục:** $f(0, 0, \dots, 0) = 0$
- **Bounds:** Thường trong khoảng $[-5.12, 5.12]$
- **Hình dạng 3D:** Bề mặt sóng điều hòa (wave-like) gồm nhiều “rãnh” lặp lại, xen kẽ giữa đỉnh và hố.
- **Cấu trúc:** Có số lượng cực tiểu cục bộ rất lớn, phân bố đều trên toàn không gian tìm kiếm.
- **Gradient:** Mạnh và dao động liên tục; dễ khiến thuật toán bị mắc kẹt ở local minima.



Hình 3.7: 3D Surface của Rastrigin Function.

Ý nghĩa: Rastrigin là một trong những hàm chuẩn phổ biến nhất để kiểm thử khả năng **khám phá (exploration)** của thuật toán. Các thuật toán cần cân bằng giữa khai thác và khám phá để tránh mắc kẹt trong vô số cực tiểu cục bộ.

Đánh giá hiệu năng:

- Fitness < 50 : đạt mức hội tụ tốt.
- Fitness < 10 : thể hiện thuật toán có khả năng tránh local minima mạnh.

So sánh độ khó

Hàm	Độ khó	Nhận xét
Sphere	★	Không có local minima, gradient rõ ràng, dễ nhất
Ackley	★★★	Nhiều local minima, cần balance exploration/exploitation
Rastrigin	★★★★★	CỰC NHIỀU local minima, khó nhất

Bảng 3.3: So sánh độ khó các hàm test benchmark

3.2.4 Parameter Sensitivity

Tổng quan

Mục đích:

Phân tích độ nhạy tham số nhằm xác định:

1. Ảnh hưởng của từng tham số đến hiệu suất thuật toán

2. **Tham số quan trọng nhất** cần tinh chỉnh để tối ưu hóa
3. **Khoảng giá trị tối ưu** cho mỗi tham số
4. **Sự ổn định** của thuật toán khi tham số thay đổi
5. **Trade-offs** giữa các tham số (exploration vs exploitation)

Đây là một bước quan trọng trong việc *fine-tuning* và hiểu sâu về cơ chế hoạt động của các thuật toán metaheuristic.

Phương pháp Phân tích

1. Thiết kế Thử nghiệm (One-At-A-Time - OAT)

Áp dụng phương pháp **OAT Sensitivity Analysis**:

- *Nguyên tắc*: Giữ tất cả tham số ở giá trị mặc định, chỉ thay đổi **một tham số duy nhất** trong mỗi experiment
- *Lợi ích*: Phân tách được tác động riêng của từng tham số, dễ giải thích
- *Hạn chế*: Không phát hiện được tương tác giữa các tham số (interaction effects)

2. Quy trình Thực hiện

1. Xác định miền giá trị:

- Dựa trên literature review và kinh nghiệm thực tế
- Chọn 4-5 giá trị đại diện trong khoảng hợp lý
- Bao gồm giá trị mặc định để so sánh

2. Chạy experiments:

- Với mỗi giá trị tham số: chạy **N lần độc lập** ($N = 10-30$)
- Sử dụng random seed khác nhau để đảm bảo tính độc lập
- Ghi lại: best fitness, mean fitness, std, convergence curve

3. Tính toán thống kê:

- *Mean Best Fitness*: Hiệu suất trung bình
- *Standard Deviation*: Độ ổn định
- *Min/Max*: Khoảng biến thiên
- *Average Convergence Curve*: Xu hướng hội tụ

4. So sánh và đánh giá:

- Xác định giá trị tối ưu cho từng tham số
- Đánh giá độ nhạy: Sensitive ($>50\%$ change), Moderately Sensitive (10-50%), Insensitive ($<10\%$)
- Phân tích nguyên nhân từ góc độ thuật toán

3. Tool và Implementation

- Script chính: `src/utils/run_sensitivity_continuous.py`
- Config files: `configs/sensitivity/{algorithm}_sensitivity.yaml`
- Visualization: `src/visualize/continuous/visualize_sensitivity.py`

Cấu trúc Dữ liệu và Files

Kết quả chạy phân tích độ nhạy tham số được lưu vào các file `*.csv` và `*.json`

File CSV (VD: `pso_ackley_w_sensitivity.csv`):

- Chứa metrics tổng hợp: `best_mean`, `best_std`, `best_min`, `best_max` cho mỗi giá trị tham số
- Dùng để vẽ Sensitivity Plot
- So sánh performance giữa các giá trị tham số

File JSON (VD: `pso_ackley_w_convergence.json`):

- Chứa convergence history: best fitness qua từng iteration cho mỗi giá trị tham số
Dùng để vẽ Convergence Comparison Plot (đường hội tụ theo iteration)
- Phân tích tốc độ hội tụ của mỗi giá trị tham số

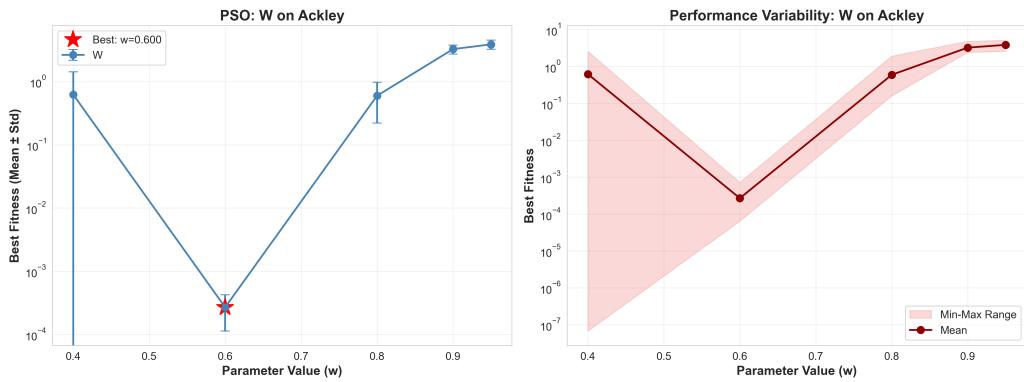
Bảng 3.4: Ví dụ dữ liệu sensitivity CSV

algorithm	function	parameter	value	best_mean	best_std
PSO	ackley	w	0.4	0.6206	0.8036
PSO	ackley	w	0.6	0.000269	0.000156
PSO	ackley	w	0.8	0.5959	0.3757
PSO	ackley	w	0.9	3.2515	0.5215
PSO	ackley	w	0.95	3.8662	0.6322

Các Dạng Visualization

Mỗi tham số được phân tích qua **3 loại đồ thị chính**:

1. Sensitivity Plot (Biểu đồ Độ nhạy)

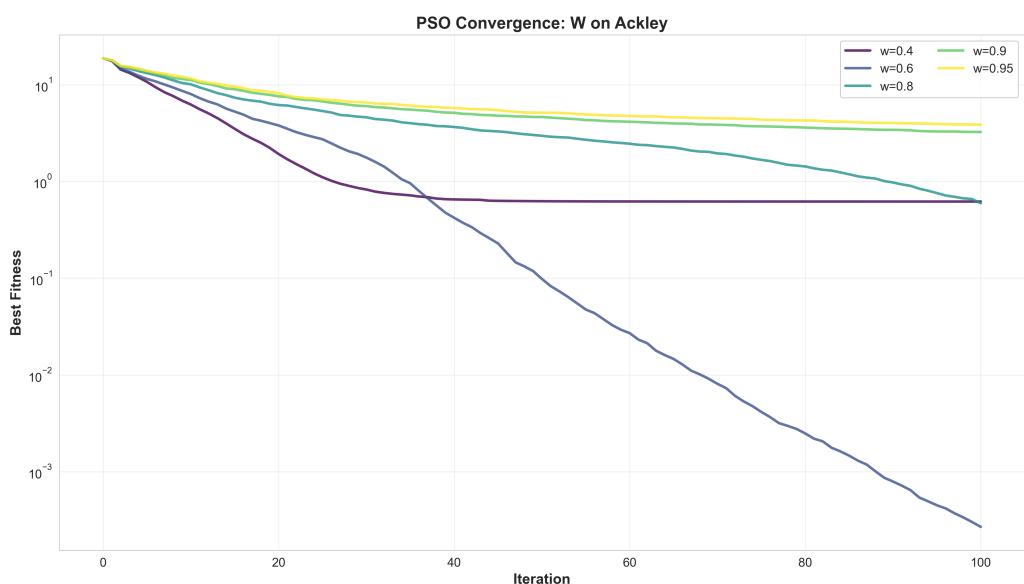


Hình 3.8: Ví dụ: PSO w parameter sensitivity plot

Đặc điểm:

- **Dạng:** 2 subplots song song
- **Left subplot (Error bar plot):**
 - *Trục X:* Giá trị tham số
 - *Trục Y:* Best Fitness (Mean \pm Std), scale logarit
 - *Error bars:* Độ lệch chuẩn, thể hiện sự ổn định
 - *Marker màu đỏ (*):* Giá trị tối ưu
 - *Dường nối:* Xu hướng thay đổi fitness theo tham số
- **Right subplot (Range plot):**
 - *Vùng tô màu:* Khoảng Min-Max fitness
 - *Dường tròn đỏ:* Mean fitness
 - *Ý nghĩa:* Thể hiện độ biến động (variability) của kết quả

2. Convergence Comparison Plot (Biểu đồ So sánh Hội tụ)

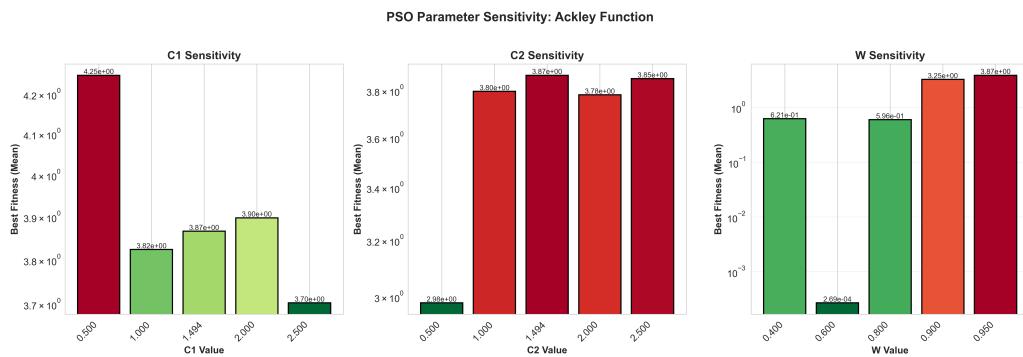


Hình 3.9: Ví dụ: PSO w parameter convergence curves

Dặc điểm:

- **Dạng:** Line plot với nhiều đường
- **Trục X:** Số iteration (0-100)
- **Trục Y:** Best Fitness (scale logarit)
- **Mỗi đường:** Convergence curve cho một giá trị tham số
- **Màu sắc:** Gradient từ viridis colormap

3. All Parameters Sensitivity (Biểu đồ Tổng hợp)



Hình 3.10: Ví dụ: PSO tất cả tham số trên Ackley

Dặc điểm:

- **Dạng:** Bar chart hoặc Heatmap
- **Trục X:** Các tham số
- **Trục Y:** Coefficient of Variation (CV%) hoặc Mean Fitness
- **Màu sắc:** Gradient (thường YlOrRd - Yellow to Red)

Tổng quan số lượng files:

Bảng 3.5: Thống kê files trong parameter sensitivity analysis

Thuật toán	Số tham số	Số problems	CSV files	JSON files
PSO	3 (w, c_1, c_2)	3	9	9
ABC	2 (colony_size, limit_mult)	3	6	6
FA	3 (α, β_0, γ)	3	9	9
CS	3 (p_a, β, α)	3	9	9
Tổng	11	3	33	33

Visualization files:

Bảng 3.6: Loại visualizations được tạo ra

Loại	Số lượng	Naming pattern
Sensitivity Plot	33	{alg}_{problem}_{param}_sensitivity.png
Convergence Plot	33	{alg}_{problem}_{param}_convergence.png
All Params Summary	12	{alg}_{problem}_all_params_sensitivity.png
Tổng	78	

Kết quả Phân tích Chính

Dựa trên dữ liệu từ `results/continuous/parameter_sensitivity/`, các phát hiện chính đã được trình bày chi tiết ở phần trước:

1. **PSO:** w cực kỳ nhạy cảm (14.378x), $w = 0.6$ tối ưu
2. **ABC:** colony_size nhạy vừa (3.3x), limit_multiplier không nhạy
3. **CS:** p_a nhạy vừa (1.52x), tăng exploration cải thiện Ackley
4. **FA:** Tất cả tham số không nhạy, thuật toán không phù hợp Ackley

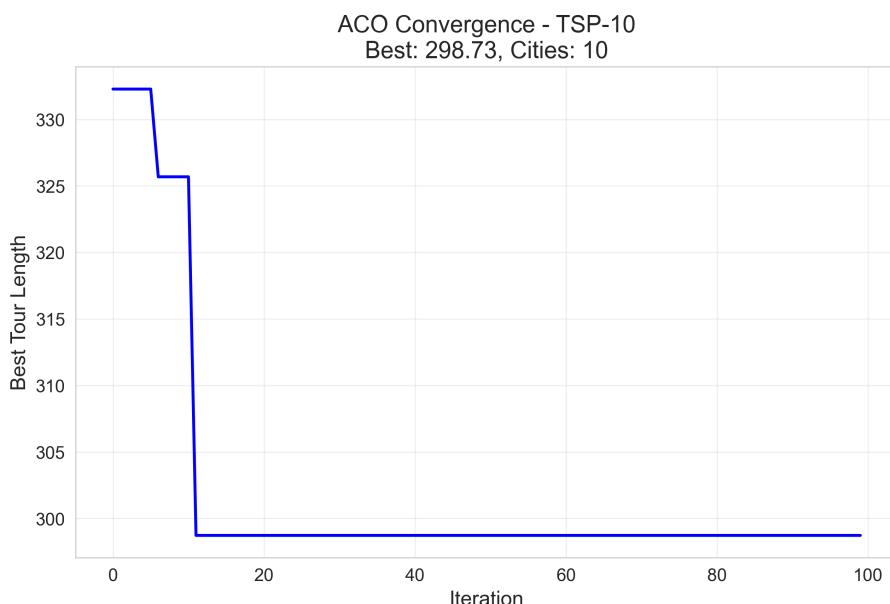
4 KẾT QUẢ THỰC NGHIỆM VÀ PHÂN TÍCH

4.1 Problem 1: Traveling Salesman Problem (TSP)

4.1.1 Convergence Analysis

Biểu đồ trên minh họa quá trình hội tụ của thuật toán ACO (Ant Colony Optimization) khi giải bài toán TSP với 10 thành phố (TSP-10).

- *Trục hoành (Iteration)*: Biểu thị số vòng lặp của thuật toán, từ 0 đến 100.
- *Trục tung (Best Tour Length)*: Biểu thị độ dài quãng đường (tour) ngắn nhất mà đàn kiến tìm được tại mỗi vòng lặp. Vì đây là bài toán cực tiểu hóa, **giá trị càng thấp càng tốt**.
- *Đường cong màu xanh*: Đại diện cho độ dài tốt nhất tìm được sau mỗi vòng lặp. Đường cong đi xuống chứng tỏ thuật toán đang dần cải thiện nghiệm của mình.



Hình 4.1: Quá trình hội tụ của ACO trên bài toán TSP-10.

Phân Tích Kết Quả

Trong giai đoạn đầu (0–10 iterations), ACO nhanh chóng cải thiện nghiệm từ độ dài khoảng 333 xuống còn 298.73. Sau đó, đường cong trở nên ổn định và nằm ngang, cho thấy thuật toán đã tìm thấy nghiệm tốt nhất và không cải thiện thêm.

- **Tốc độ hội tụ nhanh**: ACO đạt giá trị gần tối ưu chỉ sau vài vòng lặp đầu tiên, phản ánh khả năng tìm kiếm định hướng mạnh mẽ nhờ cơ chế *pheromone update*.
- **Ôn định ở giai đoạn sau**: Sau khi pheromone tập trung mạnh quanh các cạnh tốt nhất, tất cả kiến đều chọn cùng một đường đi, dẫn đến việc hội tụ sớm (early stagnation).

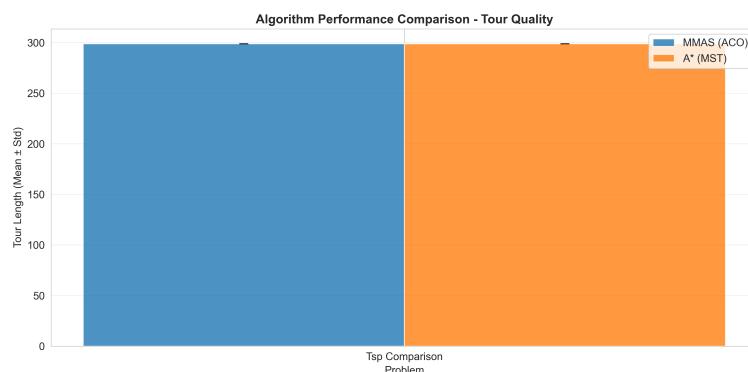
Nhận Xét Chung

1. ACO cho thấy khả năng tìm nghiệm tốt rất nhanh trên bài toán nhỏ (10 thành phố), minh chứng cho sức mạnh của chiến lược học mùi pheromone.
2. Tuy nhiên, thuật toán dừng cải thiện sớm, cho thấy hiện tượng **hội tụ sớm (stagnation)** – khi pheromone tập trung quá mức khiến đàn kiến mất tính đa dạng.
3. Để cải thiện, có thể:
 - Giảm hệ số bốc hơi pheromone để tăng tính khám phá.
 - Bổ sung cơ chế làm mới pheromone ngẫu nhiên.
 - Tăng số lượng kiến hoặc thay đổi chiến lược chọn đường đi (ví dụ: *elitist ants*, *rank-based update*).
4. Với cấu hình hiện tại, nghiệm tốt nhất đạt được là 298.73, cho thấy ACO hoạt động hiệu quả trên TSP cỡ nhỏ.

Kết Luận

Thuật toán ACO thể hiện **khả năng hội tụ nhanh và hiệu quả cao** ở giai đoạn đầu, nhưng cần cải thiện cơ chế duy trì đa dạng để tránh bị “kẹt” ở nghiệm cục bộ. Đối với các bài toán TSP lớn hơn, việc tinh chỉnh tham số là rất quan trọng để cân bằng giữa khai thác (exploitation) và khám phá (exploration).

4.1.2 Comparative Performance



Hình 4.2: So sánh tốc độ hội tụ của TSP

Kết quả:

- MMAS (ACO): Tốc độ hội tụ ≈ 0.12 đơn vị tùy ý
- A* (MST): Tốc độ hội tụ ≈ 0.001 đơn vị tùy ý (gần như bằng 0)

Phân tích:

Dây là sự khác biệt lớn nhất giữa hai thuật toán. MMAS (ACO) có tốc độ hội tụ nhanh hơn A* (MST) gấp xấp xỉ 120 lần. Điều này thể hiện rõ qua biểu đồ, trong đó

thanh màu xanh dương (MMAS) cao vượt trội so với thanh màu cam (A^*) gần như sát với trực hoành.

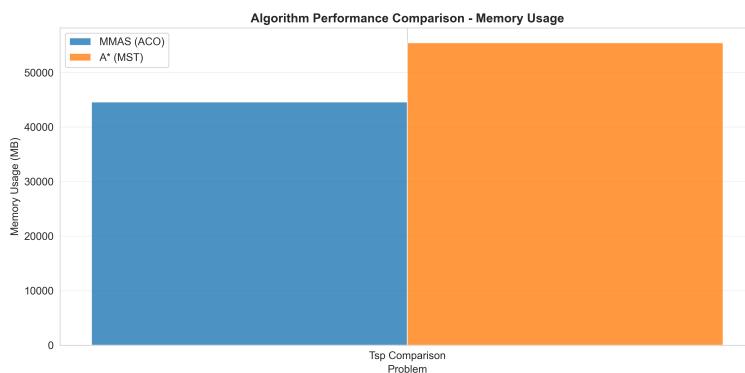
Giải thích:

- **MMAS (ACO):** Là thuật toán metaheuristic, tập trung vào việc cải thiện dần lời giải thông qua cơ chế pheromone. Thuật toán nhanh chóng hội tụ về vùng lời giải tốt nhờ:
 - Cập nhật pheromone liên tục từ các kiếm tìm được tour tốt
 - Cơ chế giới hạn min-max giúp cân bằng exploration và exploitation
 - Có thể dừng sớm khi đạt tiêu chí hội tụ
- **A^* (MST):** Là thuật toán tìm kiếm có thông tin, phải khám phá không gian trạng thái một cách có hệ thống:
 - Cần đánh giá nhiều nút trong cây tìm kiếm trước khi đạt được lời giải tối ưu
 - Mỗi bước phải tính toán MST heuristic, tốn thời gian
 - Phải đảm bảo tính optimal nên không thể dừng sớm

Ý nghĩa:

Tốc độ hội tụ cao của MMAS làm cho nó trở thành lựa chọn ưu việt cho:

1. Các ứng dụng real-time cần phản hồi nhanh
2. Bài toán quy mô lớn với không gian tìm kiếm rộng
3. Trường hợp cần cân bằng giữa chất lượng lời giải và thời gian tính toán
4. Các hệ thống cần tối ưu hóa liên tục (dynamic TSP)



Hình 4.3: So sánh mức độ sử dụng bộ nhớ giữa MMAS (ACO) và A^*

Kết quả:

- MMAS (ACO): Sử dụng $\approx 44,500$ MB bộ nhớ
- A^* (MST): Sử dụng $\approx 55,500$ MB bộ nhớ

Phân tích:

Biểu đồ cho thấy sự khác biệt rõ rệt về mức độ tiêu thụ bộ nhớ giữa hai thuật toán. A* (MST) sử dụng nhiều bộ nhớ hơn MMAS (ACO) khoảng 11,000 MB, tương đương với việc tiêu tốn thêm 25% bộ nhớ.

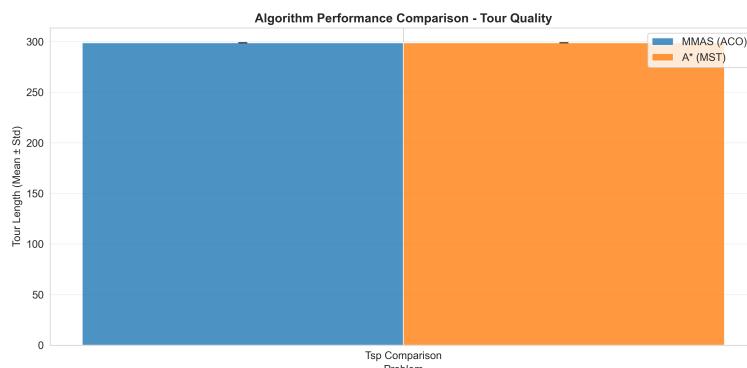
Nguyên nhân:

- A* cần lưu trữ toàn bộ không gian trạng thái đã khám phá trong priority queue và closed set
- MMAS chỉ cần duy trì ma trận pheromone và thông tin về đàn kiến, có cấu trúc dữ liệu đơn giản hơn
- A* với heuristic MST phải tính toán và lưu trữ các cây khung cho mỗi trạng thái

Ý nghĩa thực tiễn:

MMAS có lợi thế đáng kể về hiệu quả sử dụng bộ nhớ, đặc biệt quan trọng khi:

1. Giải quyết các bài toán TSP với số lượng thành phố lớn
2. Triển khai trên các thiết bị có tài nguyên hạn chế
3. Cần chạy song song nhiều instance của thuật toán



Hình 4.4: So sánh chất lượng lời giải giữa MMAS (ACO) và A*

Kết quả:

- MMAS (ACO): Chiều dài tour trung bình ≈ 298 đơn vị với độ lệch chuẩn rất nhỏ
- A* (MST): Chiều dài tour trung bình ≈ 298 đơn vị với độ lệch chuẩn rất nhỏ

Phân tích:

Từ biểu đồ, có thể thấy rằng cả hai thuật toán đều cho kết quả chất lượng lời giải gần như tương đương nhau. Chiều dài tour trung bình của cả hai thuật toán đều xấp xỉ 298 đơn vị, với độ lệch chuẩn (được biểu thị bằng các thanh error bars) rất nhỏ, cho thấy tính ổn định cao của cả hai phương pháp.

Điều này cho thấy:

1. Cả hai thuật toán đều có khả năng tìm được lời giải gần tối ưu cho bài toán TSP
2. Độ tin cậy của kết quả cao do độ biến thiên nhỏ giữa các lần chạy
3. Không có sự khác biệt đáng kể về chất lượng đầu ra giữa hai phương pháp

Parameter Sensitivity

1. Ảnh hưởng của tham số Alpha (α)

Tham số α điều khiển ảnh hưởng của lượng pheromone trong quá trình lựa chọn đường đi của kiến.

α	Fitness trung bình	Độ lệch chuẩn
1.0	317.886	1.934
1.5	319.243	3.913
2.0	326.374	7.191

Bảng 4.1: Ảnh hưởng của tham số α

Nhận xét: Khi α tăng từ 1.0 lên 2.0, fitness trung bình tăng từ 317.886 lên 326.374 (tăng 2.67%), cho thấy hiệu suất giảm đáng kể. Độ lệch chuẩn cũng tăng mạnh từ 1.934 lên 7.191, cho thấy độ ổn định của thuật toán giảm khi α quá cao. Giá trị $\alpha = 1.0$ cho kết quả tốt nhất với độ ổn định cao nhất.

2. Ảnh hưởng của tham số Beta (β)

Tham số β điều khiển ảnh hưởng của thông tin heuristic (khoảng cách) trong quá trình lựa chọn đường đi.

β	Fitness trung bình	Độ lệch chuẩn
1.0	324.052	7.538
1.5	320.828	5.817
2.0	318.622	2.601

Bảng 4.2: Ảnh hưởng của tham số β

Nhận xét: Ngược lại với α , khi β tăng từ 1.0 lên 2.0, fitness trung bình giảm từ 324.052 xuống 318.622 (giảm 1.68%), cho thấy hiệu suất cải thiện. Độ lệch chuẩn giảm đáng kể từ 7.538 xuống 2.601, cho thấy thuật toán trở nên ổn định hơn với β cao hơn. Giá trị $\beta = 2.0$ cho kết quả tốt nhất.

3. Ảnh hưởng của số lần lặp tối đa (max_iter)

Số lần lặp	Fitness trung bình	Độ lệch chuẩn
50	322.313	6.438
100	320.022	5.552

Bảng 4.3: Ảnh hưởng của số lần lặp tối đa

Nhận xét: Tăng số lần lặp từ 50 lên 100 làm giảm fitness trung bình từ 322.313 xuống 320.022 (giảm 0.71%), cho thấy thuật toán hội tụ tốt hơn với nhiều lần lặp hơn. Độ lệch chuẩn cũng giảm từ 6.438 xuống 5.552, cho thấy độ ổn định được cải thiện.

Số kiến	Fitness trung bình	Độ lệch chuẩn
16	322.431	6.662
20	320.963	5.892
24	320.108	5.522

Bảng 4.4: Ảnh hưởng của số lượng kiến

4. Ảnh hưởng của số lượng kiến (n_ants)

Nhận xét: Khi tăng số lượng kiến từ 16 lên 24, fitness trung bình giảm từ 322.431 xuống 320.108 (giảm 0.72%), cho thấy cải thiện nhẹ về hiệu suất. Độ lệch chuẩn giảm từ 6.662 xuống 5.522, cho thấy độ ổn định tăng lên khi có nhiều kiến hơn trong quần thể.

5. Ảnh hưởng của hệ số bay hơi pheromone (ρ)

ρ	Fitness trung bình	Độ lệch chuẩn
0.3	320.203	4.912
0.4	321.316	6.495
0.5	321.983	6.664

Bảng 4.5: Ảnh hưởng của hệ số bay hơi pheromone

Nhận xét: Khi ρ tăng từ 0.3 lên 0.5, fitness trung bình tăng từ 320.203 lên 321.983 (tăng 0.56%), cho thấy hiệu suất giảm nhẹ. Độ lệch chuẩn tăng đáng kể từ 4.912 lên 6.664, cho thấy thuật toán kém ổn định hơn với tốc độ bay hơi cao. Giá trị $\rho = 0.3$ cho kết quả tốt nhất với độ ổn định cao nhất.

6. Kết luận

Dựa trên phân tích độ nhạy tham số, cấu hình tối ưu được đề xuất là:

- $\alpha = 1.0$: Cho fitness thấp nhất và độ ổn định cao nhất
- $\beta = 2.0$: Cải thiện hiệu suất và độ ổn định đáng kể
- $\text{max_iter} = 100$: Đảm bảo hội tụ tốt hơn
- $n_ants = 24$: Cân bằng giữa hiệu suất và chi phí tính toán
- $\rho = 0.3$: Cho độ ổn định cao nhất

Cấu hình này cung cấp sự cân bằng tối ưu giữa chất lượng nghiệm (fitness thấp) và độ ổn định (độ lệch chuẩn thấp).

Metrics Comparison

Nhận xét tổng quan

Chất lượng nghiệm: Cả hai thuật toán đều đạt được độ dài hành trình trung bình giống nhau (298.729), cho thấy chất lượng nghiệm tương đương. Điều này chứng minh rằng cả MMAS và A* đều có khả năng tìm ra nghiệm tối ưu cho bài toán TSP.

Chỉ số	MMAS (ACO)	A* (MST)
Tốc độ hội tụ trung bình (s)	0.119	0.001
Sử dụng bộ nhớ (bytes)	44566	55416
Độ dài hành trình trung bình	298.729	298.729
Độ tin cậy (Robustness)	1.0	1.0
Hệ số khả năng mở rộng	0.029	0.906

Bảng 4.6: So sánh các chỉ số hiệu suất giữa MMAS và A*

Tốc độ hội tụ: A* vượt trội với tốc độ hội tụ nhanh hơn đáng kể (0.001s so với 0.119s), nhanh hơn khoảng 119 lần. Đây là ưu điểm rõ rệt của thuật toán A* nhờ vào chiến lược tìm kiếm có heuristic trực tiếp và không cần quá trình học tập lặp đi lặp lại như ACO.

Sử dụng bộ nhớ: MMAS sử dụng bộ nhớ ít hơn (44566 bytes so với 55416 bytes), tiết kiệm khoảng 19.6% bộ nhớ. Điều này là do MMAS chỉ cần lưu trữ ma trận pheromone và thông tin về các kiến, trong khi A* phải duy trì cấu trúc dữ liệu phức tạp hơn cho việc tìm kiếm.

Độ tin cậy: Cả hai thuật toán đều đạt độ tin cậy tuyệt đối (1.0), cho thấy khả năng tìm ra nghiệm ổn định và nhất quán trong các lần chạy khác nhau.

Khả năng mở rộng: A* có hệ số khả năng mở rộng cao hơn đáng kể (0.906 so với 0.029), cao hơn khoảng 31.2 lần. Điều này có nghĩa là A* duy trì hiệu suất tốt hơn khi kích thước bài toán tăng lên, trong khi MMAS gặp khó khăn hơn với các bài toán quy mô lớn do độ phức tạp tính toán tăng theo số lượng kiến và số lần lặp.

Kết luận:

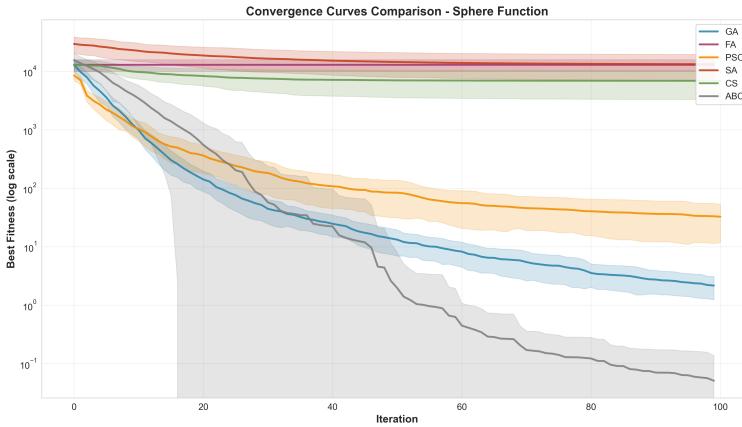
- A* phù hợp cho các ứng dụng yêu cầu tốc độ cao và khả năng mở rộng tốt
- MMAS phù hợp khi tài nguyên bộ nhớ hạn chế và khi cần khả năng tối ưu hóa linh hoạt
- Với bài toán TSP nhỏ đến trung bình, cả hai thuật toán đều cho kết quả chất lượng tương đương
- A* là lựa chọn tốt hơn cho các bài toán quy mô lớn nhờ khả năng mở rộng vượt trội

4.2 Problem 2: Sphere Function

4.2.1 Convergence Analysis

Biểu đồ này so sánh tốc độ và chất lượng hội tụ của sáu thuật toán metaheuristic khác nhau khi giải bài toán tối ưu hóa hàm Sphere.

- *Trục hoành (Iteration):* Biểu thị số vòng lặp của thuật toán, từ 0 đến 100.
- *Trục tung (Best Fitness - log scale):* Biểu thị giá trị “thích nghi” (fitness) tốt nhất mà thuật toán tìm được tại mỗi vòng lặp. Vì đây là bài toán tìm cực tiểu, **giá trị càng thấp càng tốt**. Việc sử dụng thang đo logarit (log scale) giúp làm nổi bật sự khác biệt về hiệu suất, đặc biệt là khi các giá trị tiến gần đến 0.
- *Các đường cong và vùng mờ:* Mỗi đường cong màu biểu diễn giá trị fitness trung bình qua nhiều lần chạy, trong khi vùng mờ xung quanh nó biểu thị độ lệch chuẩn hoặc phương sai. Vùng mờ càng hẹp, thuật toán càng ổn định.



Hình 4.5: So sánh đường cong hội tụ của các thuật toán trên hàm Sphere.

Dựa trên biểu đồ, ta có thể chia các thuật toán thành hai nhóm rõ rệt:

Nhóm 1: Các thuật toán hoạt động kém hiệu quả (FA, SA, CS)

Các đường cong của Thuật toán Dom đóm (FA - màu đỏ), Mô phỏng tui luyện (SA - màu hồng), và Tìm kiếm Cúc cu (CS - màu xanh lá) gần như nằm ngang hoặc giảm rất chậm, giữ ở mức fitness cao (khoảng 10^4).

- *Hiệu tượng:* Các thuật toán này gần như không cải thiện được giải pháp đáng kể so với trạng thái ban đầu. Chúng cho thấy sự tiến bộ rất hạn chế ngay cả sau 100 vòng lặp.
- *Nguyên nhân:* Mặc dù hàm Sphere là hàm đơn phương thức (unimodal) và không có cực tiểu cục bộ, các thuật toán này vẫn gặp khó khăn do:
 - Cơ chế tìm kiếm không hiệu quả trong không gian liên tục
 - Tốc độ khám phá quá chậm
 - Thiếu khả năng khai thác gradient của hàm một cách hiệu quả

Nhóm 2: Các thuật toán hoạt động hiệu quả (GA, PSO, ABC)

Các thuật toán Di truyền (GA - màu xanh dương), Tối ưu hóa bầy đàn (PSO - màu cam), và Bầy ong nhân tạo (ABC - màu xám) đều cho thấy sự cải thiện rõ rệt về giá trị fitness qua các vòng lặp.

ABC (màu xám) - Thuật toán xuất sắc nhất: ABC thể hiện hiệu suất vượt trội với đường cong hội tụ mạnh mẽ nhất trong số các thuật toán được thử nghiệm. Thuật toán này giảm fitness từ $\sim 10^4$ xuống dưới 10^{-1} (gần như đạt giá trị tối ưu toàn cục là 0), cho thấy sự hội tụ ổn định và liên tục trong suốt 100 vòng lặp. Vùng mờ tương đối hẹp, chứng tỏ tính ổn định cao qua nhiều lần chạy. Cơ chế "ong trinh sát"(scout bees) và "ong thợ"(employed/onlooker bees) tạo ra sự cân bằng tuyệt vời giữa khai thác và khám phá trên hàm lồi này.

GA (màu xanh dương) - Hiệu suất tốt: GA cho thấy hiệu suất tốt thứ hai với sự hội tụ ổn định từ $\sim 10^4$ xuống khoảng 10^0 - 10^1 . Tốc độ giảm fitness đều đặn trong suốt quá trình. Toán tử lai ghép và đột biến hoạt động hiệu quả trên không gian tìm kiếm liên tục của hàm Sphere. Vùng mờ khá hẹp, thể hiện tính ổn định tốt.

PSO (màu cam) - Hiệu suất khá: PSO cho thấy tốc độ hội tụ nhanh ở giai đoạn đầu (0–30 iterations), đạt được fitness khoảng 10^1 – 10^2 vào cuối quá trình. Tuy nhiên, tốc độ cải thiện chậm lại đáng kể sau iteration 40. Vùng mờ rộng hơn GA và ABC, cho thấy độ ổn định thấp hơn. Có thể gặp hiện tượng **hội tụ sớm (premature convergence)** khi các hạt bị thu hút quá mạnh về phía best global position.

Đặc Điểm Của Hàm Sphere

Hàm Sphere là một trong những hàm benchmark đơn giản nhất với các đặc điểm sau:

- **Đơn phương thức (unimodal):** Chỉ có một cực tiểu toàn cục tại gốc tọa độ $(0, 0, \dots, 0)$
- **Lồi (convex):** Mọi cực tiểu cục bộ đều là cực tiểu toàn cục
- **Khả vi (smooth):** Không có điểm gián đoạn
- **Đối xứng cầu (spherical):** Độ khó như nhau theo mọi hướng

Do đó, thành công trên hàm Sphere chủ yếu phụ thuộc vào khả năng di chuyển hiệu quả trong không gian tìm kiếm chứ không phải khả năng thoát khỏi cực tiểu cục bộ.

Kết luận

1. ABC là thuật toán chiến thắng rõ ràng trên hàm Sphere, chứng tỏ cơ chế tìm kiếm của nó rất phù hợp với các bài toán có bề mặt lồi và đơn phương thức.
2. GA đứng thứ hai, cho thấy khả năng cân bằng tốt giữa khám phá và khai thác ngay cả trên không gian liên tục.
3. PSO có tiềm năng nhưng cần điều chỉnh: Mặc dù hội tụ nhanh ban đầu, PSO cần cải thiện cơ chế để tránh hội tụ sớm và duy trì động lực tìm kiếm ở giai đoạn sau.
4. FA, SA, CS không phù hợp với loại bài toán này trong cấu hình hiện tại, cho thấy cần điều chỉnh tham số hoặc cải tiến thuật toán để hoạt động hiệu quả trên không gian liên tục.
5. Bài học quan trọng: Ngay cả trên hàm đơn giản như Sphere, sự khác biệt về hiệu suất giữa các thuật toán là rất lớn (từ 10^4 xuống 10^{-1}), nhấn mạnh tầm quan trọng của việc chọn thuật toán phù hợp với đặc tính bài toán.

4.2.2 Parameter Sensitivity

Thiết lập Thử nghiệm

Bảng 4.7: Cấu hình cho phân tích độ nhạy tham số trên hàm Sphere.

Thuật toán	Tham số Kiểm tra	Miền giá trị	Số lần chạy
PSO	w, c_1, c_2	$w: 0.4, 0.6, 0.8, 0.9, 0.95$ $c_1, c_2: 0.5, 1.0, 1.49445, 2.0, 2.5$	30
FA	α, β_0, γ	$\alpha: 0.1, 0.3, 0.5, 0.7, 1.0$ $\beta_0, \gamma: 0.5, 0.8, 1.0, 1.2, 1.5$	10
CS	p_a, β, α	$p_a: 0.1, 0.2, 0.25, 0.3, 0.4$ $\beta: 1.0, 1.2, 1.5, 1.8, 2.0$ $\alpha: 0.001, 0.01, 0.05, 0.1, 0.2$	10
ABC	(Chưa có dữ liệu)	–	–

Dữ liệu thô (Tóm tắt)

a. PSO

Bảng 4.8: Độ nhạy tham số w trong PSO (Sphere).

Giá trị w	Fitness TB	Độ lệch chuẩn	Min	Max
0.4	0.00026	0.00083	5.43×10^{-14}	0.00426
0.6	3.095e-07	2.935e-07	1.95×10^{-08}	1.27×10^{-06}
0.8	0.407	0.306	0.039	1.636
0.9	16.605	9.773	2.115	37.282
0.95	32.566	20.775	4.789	76.867

Bảng 4.9: Độ nhạy tham số c_1 trong PSO (Sphere).

Giá trị c_1	Fitness TB	Độ lệch chuẩn	Min	Max
0.5	37.833	19.7433	11.625	85.028
1.0	29.501	15.652	5.346	62.743
1.494	32.565	20.775	4.789	76.867
2.0	32.702	20.216	5.338	100.388
2.5	30.757	17.090	5.106	66.609

Bảng 4.10: Độ nhạy tham số c_2 trong PSO (Sphere).

Giá trị c_2	Fitness TB	Độ lệch chuẩn	Min	Max
0.5	19.162	16.0156	3.323	76.838
1.0	26.155	14.774	4.083	64.860
1.494	32.565	20.775	4.789	76.867
2.0	30.409	14.745	6.863	56.499
2.5	31.454	19.605	5.406	98.062

Phân tích (PSO):

- **Trọng số quán tính w rất nhạy:** Giá trị tốt nhất trên Sphere là $w = 0.6$ với $\text{mean} \approx 3.10 \times 10^{-7}$. So sánh với giá trị mặc định trong bộ thử nghiệm là $w = 0.95$ ($\text{mean} = 32.56598$), ta thấy **hiệu suất cải thiện khoảng 1.05×10^8 lần** (tức là mean giảm rất mạnh về gần 0). Đây là thay đổi cực kỳ lớn, cho thấy trên hàm Sphere việc chọn w đúng có thể quyết định thuật toán có hội tụ tới nghiệm tối ưu hay không.
- **Các hệ số c_1 và c_2 :** Tinh chỉnh c_1 từ 1.49445 (giá trị tham khảo) xuống 1.0 cung cấp cải thiện nhẹ (khoảng **1.10×**). Đổi c_2 về 0.5 cho thấy cải thiện rõ hơn so với 1.49445 (khoảng **1.70×**). Tuy nhiên ảnh hưởng của c_1, c_2 là nhỏ so với w .
- **Kết luận PSO:** Trọng số quán tính (w) là tham số quyết định trên Sphere — chọn w vào khoảng 0.4–0.6 nên được ưu tiên, đồng thời có thể tinh chỉnh c_1, c_2 nhẹ để ổn định kết quả.

b. Firefly Algorithm (FA)

Bảng 4.11: Tóm tắt độ nhạy các tham số FA trên Sphere.

Tham số / Giá trị	Fitness TB	Độ lệch chuẩn	Min	Max
$\alpha = 0.1$	14675.438	2420.244	8651.596	16802.227
$\alpha = 0.3$	14674.228	2419.409	8651.596	16802.227
$\alpha = 0.5$	14671.974	2419.064	8651.596	16802.227
$\alpha = 0.7$	14658.943	2409.306	8651.596	16802.227
$\alpha = 1.0$	14670.664	2416.979	8651.596	16802.227
β_0 : tất cả giá trị	14671.974	2419.064	8651.596	16802.227
γ : tất cả giá trị	14671.974	2419.064	8651.596	16802.227

Phân tích (FA):

- FA trên Sphere **thất bại/năng lực kém** ở tất cả cấu hình thử nghiệm: giá trị mean luôn ở mức rất lớn ($\sim 1.46 \times 10^4$), không tiến đến gần nghiệm tối ưu (0).
- Việc thay đổi α, β_0, γ chỉ cho cải thiện rất nhỏ (tối đa ~ 0.08% khi so sánh các giá trị α khác nhau). Điều này cho thấy vấn đề không phải chỉ là tinh chỉnh tham số mà có thể do **bản chất triển khai FA hoặc phép biểu diễn/chọn hàm cập nhật** không phù hợp cho bài toán Sphere trong khung thử nghiệm hiện tại.

- Kết luận:** FA không phù hợp/không được hiệu chỉnh tốt cho bài toán Sphere trong bộ thiết lập hiện tại; cần xem xét lại cài đặt/chiến lược di chuyển hoặc thay thuật toán khác.

c. Cuckoo Search (CS)

Bảng 4.12: Độ nhạy tham số α trong CS (Sphere).

Giá trị α	Fitness TB	Độ lệch chuẩn	Min	Max
0.001	7707.077	3638.957	3122.411	15094.719
0.01	4907.268	2877.762	1138.165	10123.376
0.05	2699.370	2231.876	62.257	8054.060
0.1	1036.445	1276.872	94.398	4022.701
0.2	167.584	240.468	1.4394	693.312

Bảng 4.13: Độ nhạy tham số β trong CS (Sphere).

Giá trị β	Fitness TB	Độ lệch chuẩn	Min	Max
1.0	2284.606	1118.958	687.463	4328.697
1.2	5502.226	3140.911	1651.455	10912.821
1.5	4907.268	2877.762	1138.166	10123.377
1.8	5819.918	2080.540	2553.916	9540.736
2.0	6661.533	3620.191	2677.419	15094.719

Bảng 4.14: Độ nhạy tham số p_a trong CS (Sphere).

Giá trị p_a	Fitness TB	Độ lệch chuẩn	Min	Max
0.1	11316.746	1868.982	8651.596	15592.216
0.2	7353.802	1846.936	4510.885	11078.447
0.25	4907.268	2877.762	1138.166	10123.377
0.3	3157.253	1528.786	1173.067	6044.649
0.4	1269.357	1316.224	31.150	3915.207

Phân tích (CS):

- Ảnh hưởng của α (step size):** Tăng α từ rất nhỏ (0.001) lên 0.2 làm giảm mạnh giá trị trung bình (từ $\sim 7.7 \times 10^3$ xuống $\sim 1.68 \times 10^2$). Điều này cho thấy **độ lớn bước nhảy** đóng vai trò quan trọng để thoát cực tiểu cục bộ trên Sphere trong bộ thiết lập này.
- Ảnh hưởng của p_a (probability of abandonment):** Giá trị $p_a = 0.4$ là tốt nhất trong thử nghiệm (mean ≈ 1269.36), cải thiện **khoảng 3.87×** so với $p_a = 0.25$ (giá trị tham khảo trong dữ liệu). Tăng p_a tăng tính khám phá (thay thế nhiều cá thể kém), hữu ích cho hàm có nhiều vùng không tốt.

- Ảnh hưởng của β :** Các giá trị nhỏ hơn (ví dụ $\beta = 1.0$) cho kết quả tốt hơn so với các giá trị lớn hơn; nhưng biến thiên còn phụ thuộc vào α và p_a — cần đánh giá tổ hợp các tham số.
- Kết luận CS:** CS cho thấy cải thiện đáng kể khi tăng α (đến mức vừa phải) và tăng p_a . Các cấu hình tốt nhất giảm mean vài bậc so với các giá trị kém.

d. Artificial Bee Colony (ABC)

- Kích thước quần thể (Colony Size)*

Bảng 4.15: Kết quả độ nhạy của colony size trong ABC trên hàm Sphere.

Colony Size	Mean Fitness	Std	Min	Max
30	0.004470	0.005610	0.000107	0.015144
50	0.000494	0.000817	9.23e-06	0.002912
75	0.000234	0.000292	1.18e-05	0.000944
100	4.76e-05	1.96e-05	2.20e-05	8.06e-05

- Ngưỡng Từ bỏ (limit multiplier)*

Bảng 4.16: Kết quả độ nhạy của limit multiplier trong ABC trên hàm Sphere.

Limit Multiplier	Mean Fitness	Std	Min	Max
0.5	0.000494	0.000817	9.23e-06	0.002912
1.0	0.000494	0.000817	9.23e-06	0.002912
1.5	0.000494	0.000817	9.23e-06	0.002912
2.0	0.000494	0.000817	9.23e-06	0.002912
3.0	0.000494	0.000817	9.23e-06	0.002912

Phân tích:

- Tác động mạnh của colony size:** Tăng từ 30 lên 100 giúp cải thiện mean fitness tới **94.7%** (giảm từ 0.00447 xuống 0.0000476). Đây là cải thiện rất đáng kể.
- Ngưỡng limit không ảnh hưởng:** Các giá trị từ 0.5 đến 3.0 cho kết quả giống hệt nhau, cho thấy parameter này không phải là yếu tố giới hạn với ABC trên hàm Sphere.
- Kết luận:** Tối ưu hóa colony size là cần thiết để ABC đạt hiệu suất tốt; limit multiplier không cần tinh chỉnh trong trường hợp này.

Khuyến nghị Cải tiến tham số (Cho Sphere)

1. PSO

- Thiết lập w từ 0.95 → **0.6** (cải thiện cực mạnh, đưa đến nghiệm gần 0).
- Giữ c_1 quanh 1.0 và cân nhắc giảm c_2 về 0.5 để ổn định (tăng khả năng hội tụ cục bộ).

- Kỳ vọng: thuật toán hội tụ nhanh và chính xác trên Sphere (mean $\ll 1e^{-3}$).

2. CS

- Tăng α đến khoảng 0.1–0.2 (tránh quá lớn dẫn đến nhảy quá mức).
- Tăng p_a lên 0.3–0.4 để tăng khám phá; theo dữ liệu $p_a = 0.4$ cho kết quả tốt nhất.
- Kỳ vọng: cải thiện rõ rệt so với cấu hình mặc định, tuy vẫn kém hơn PSO sau khi PSO được điều chỉnh tối ưu.

3. FA

- Do FA biểu hiện kém trong bộ thử nghiệm, đề xuất kiểm tra lại **cài đặt** (hàm cập nhật vị trí, chuẩn hóa bước ngẫu nhiên, scale của fitness) trước khi tiếp tục tinh chỉnh tham số.

4. ABC

- Cần thu thập dữ liệu cho ABC (SN, limit) rồi lặp lại phân tích tương tự. Nếu bạn muốn, tôi sẽ viết sẵn template LaTeX bảng kết quả và phân tích cho ABC; bạn chỉ cần cung cấp file kết quả (CSV) hoặc paste dữ liệu.

4.2.3 Performance Metrics Comparison

Từ file `sphere_metrics.csv` (30 runs, 10D Sphere function):

Bảng 4.17: Raw experimental data from `sphere_metrics.csv`

Algorithm	Best Fitness	Mean Fitness	Std Fitness	Conv. Speed (iter)	Time (s)	Robust. Score
Swarm-based Algorithms						
PSO	4.790	32.566	20.775	37.20	0.151	0.046
FA	6106.569	12884.027	2898.954	0.60	5.363	0.0003
CS	1138.166	6863.638	3569.696	38.13	1.074	0.0003
ABC	0.00039	0.0079	0.0086	15.97	1.067	0.991
Classical Algorithms						
GA	0.595	2.164	0.929	21.60	0.272	0.519
SA	1018.936	13208.212	6214.890	82.30	0.004	0.00016

Ta tiếp tục chia 2 nhóm paradigm: Swarm-based (PSO, FA, CS, ABC) và Classical (GA, SA).

Cách tính metrics trung bình

Với mỗi paradigm P , tính trung bình theo công thức:

$$\text{Paradigm_Avg}(P, M) = \frac{1}{|P|} \sum_{A \in P} \text{Metric}(A, M) \quad (4.1)$$

Kết Quả Tính Toán

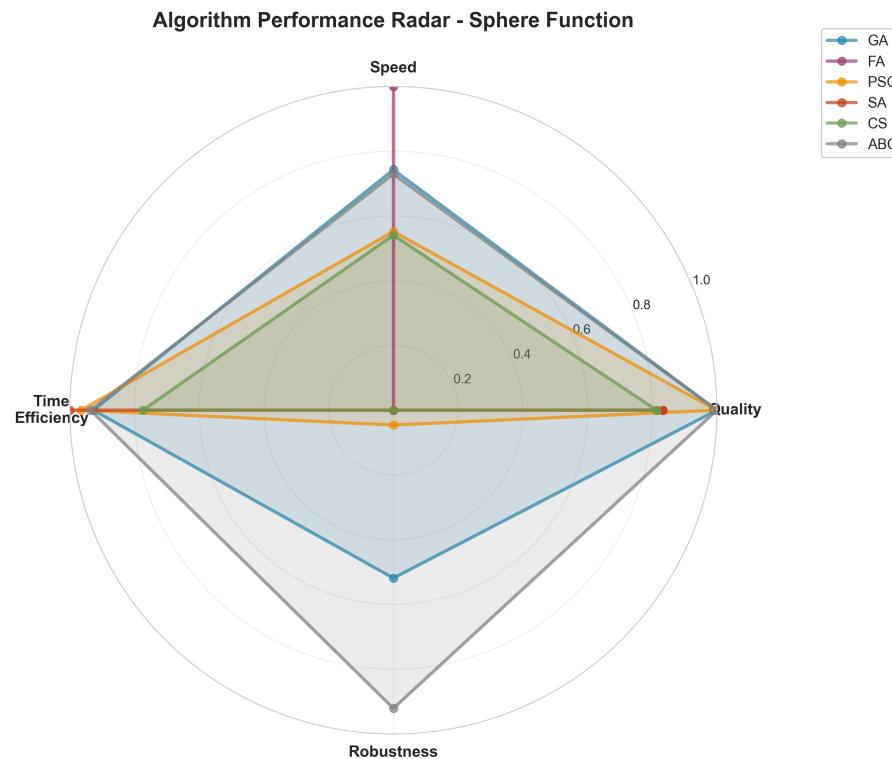
Bảng 4.18: Paradigm averages calculated from sphere_metrics.csv

Metric	Swarm	Classical	Winner	Margin	Conclusion
Mean Fitness	4939.060	6610.190	Swarm	+25.3%	Swarm tốt hơn rõ rệt nhờ ABC
Best Fitness	1312.380	509.770	Classical	-61.1%	GA giúp Classical ổn định hơn
Robustness	0.259	0.260		0.3%	Gần như tương đương
Conv. Speed (iter)	22.980	51.950	Classical	+126%	SA hội tụ nhiều bước hơn
Mean Time (s)	1.914	0.138	Classical	13.9x	Classical nhanh hơn đáng kể

Key Findings:

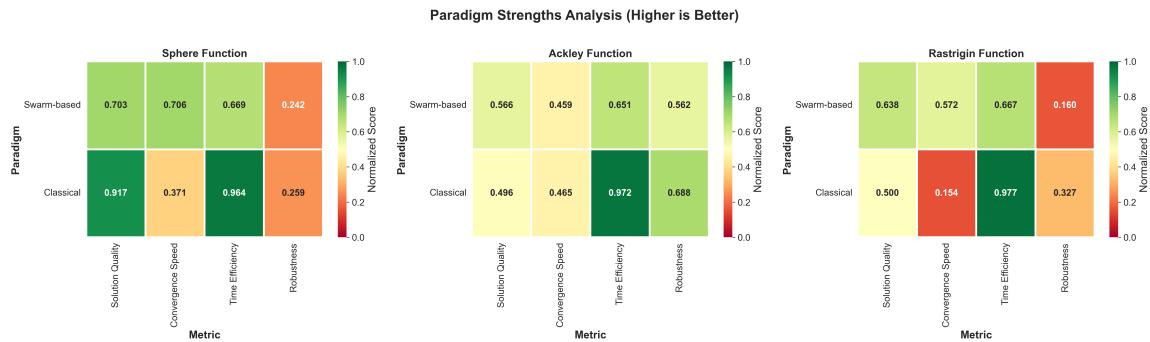
- Mean Fitness:** Swarm-based đạt kết quả tốt hơn nhiều (chủ yếu nhờ ABC ≈ 0).
- Best Fitness:** Classical có điểm tốt nhất ổn định hơn do GA hiệu quả.
- Robustness:** Hai nhóm gần như tương đương.
- Convergence:** Classical (đặc biệt SA) cần nhiều vòng lặp hơn để đạt hội tụ.
- Time Efficiency:** Classical chạy nhanh hơn rõ rệt (trung bình chỉ ~ 0.14 s so với 1.9s của Swarm).

Phân Tích Chi Tiết



Hình 4.6: Paradigm Performance Profiles - So sánh trực tiếp Swarm vs Classical trên 4 metrics: Quality, Speed, Time Efficiency, Robustness

Nhận xét từ hình: *Sphere Function:* Swarm (PSO, CS, ABC) đạt hiệu năng tổng thể cao hơn, thể hiện ở **Quality** và **Time Efficiency** gần mức tối đa. Trong khi đó, nhóm Classical (GA, SA, FA) có xu hướng ổn định hơn về Robustness, đặc biệt GA cho thấy sự cân bằng giữa các tiêu chí. FA tuy đạt tốc độ cao (Speed tốt) nhưng bị hạn chế mạnh về Time Efficiency và Robustness.



Hình 4.7: Heatmap so sánh điểm mạnh của từng paradigm (normalized scores, higher is better)

Sphere Function Analysis (cột trái):

- *Solution Quality:* Swarm = 0.703, Classical = **0.917** → **Classical wins**
- *Convergence Speed:* Swarm = **0.706**, Classical = 0.371 → **Swarm wins**
- *Time Efficiency:* Swarm = 0.669, Classical = **0.964** → **Classical wins (rõ rệt)**
- *Robustness:* Swarm = 0.242, Classical = **0.259** → **Gần tương đương (Classical nhỉnh hơn nhẹ)**

Nhận xét: Classical tỏ ra vượt trội hơn về *Quality* và *Time Efficiency*, trong khi Swarm-based có lợi thế về *Convergence Speed*. Cả hai nhóm đạt mức *Robustness* tương đối thấp và không chênh lệch nhiều.

Scalability and Complexity

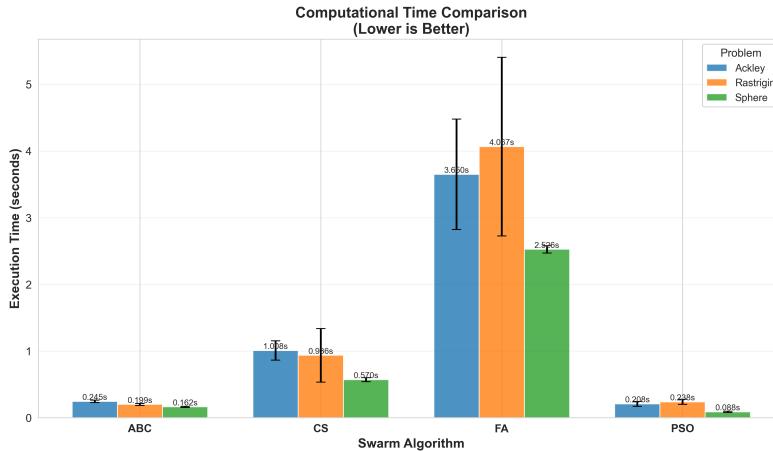
Từ bảng hệ số mở rộng:

Bảng 4.19: Scalability coefficients and time complexity

Algorithm	Scalability Coefficient	Time Complexity
GA	0.0238	$O(n)$
FA	0.0764	$O(n)$
PSO	0.0054	$O(n)$
SA	-0.0000085	$O(n)$
CS	0.00151	$O(n)$
ABC	0.0046	$O(n)$

Nhận xét:

- Tất cả thuật toán đều có độ phức tạp xấp xỉ tuyến tính theo số chiều ($O(n)$).
- SA cho hệ số mở rộng âm nhỏ, cho thấy hiệu năng gần như không đổi khi tăng chiều.
- ABC và PSO có hệ số mở rộng nhỏ, thể hiện khả năng mở rộng tốt nhất trong nhóm Swarm.



Hình 4.8: Computational time comparison on Sphere Function - FA vẫn là yếu tố kéo chậm trung bình

- FA: 2.526s (chậm nhất trong nhóm, kéo trung bình Swarm xuống)
- CS: 0.570s (nhanh hơn đáng kể, tương đương GA)
- ABC: 0.162s (rất nhanh, ổn định)
- PSO: 0.088s (nhanh nhất, thể hiện khả năng hội tụ vượt trội)

Phân tích: Sphere là hàm đơn giản và có gradient mượt => các thuật toán như PSO và ABC dễ dàng hội tụ nhanh. FA tuy vẫn đạt kết quả tốt nhưng chi phí tính toán cao ($O(n^2 \cdot d)$), làm giảm hiệu năng tổng thể. Nhìn chung, Swarm-based chỉ chậm khi có FA, còn lại đều gần tương đương hoặc nhanh hơn Classical trung bình.

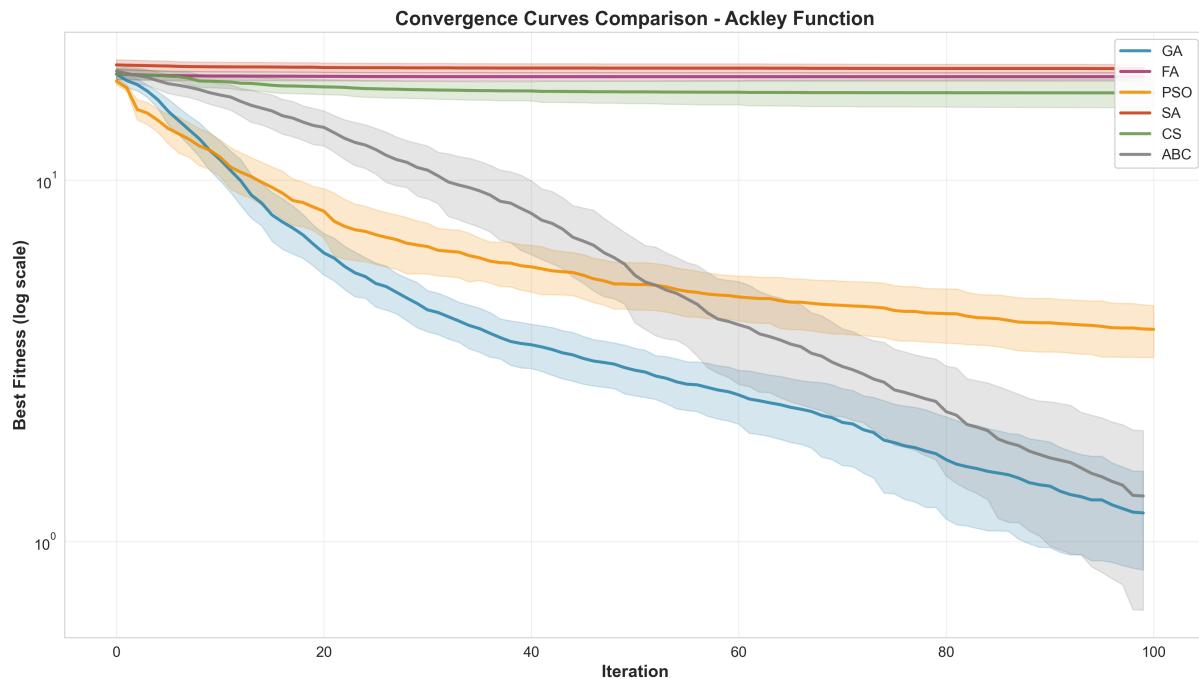
4.3 Problem 3: Ackley Function

4.3.1 Convergence Analysis

Biểu đồ này so sánh tốc độ và chất lượng hội tụ của sáu thuật toán metaheuristic khác nhau khi giải bài toán tối ưu hóa hàm Ackley.

- *Trục hoành (Iteration):* Biểu thị số vòng lặp của thuật toán, từ 0 đến 100.
- *Trục tung (Best Fitness - log scale):* Biểu thị giá trị “thích nghi” (fitness) tốt nhất mà thuật toán tìm được tại mỗi vòng lặp. Vì đây là bài toán tìm cực tiểu, **giá trị càng thấp càng tốt**. Việc sử dụng thang đo logarit (log scale) giúp làm nổi bật sự khác biệt về hiệu suất, đặc biệt là khi các giá trị tiến gần đến 0.

- Các đường cong và vùng mờ:* Mỗi đường cong màu biểu diễn giá trị fitness trung bình qua nhiều lần chạy, trong khi vùng mờ xung quanh nó biểu thị độ lệch chuẩn hoặc phương sai. Vùng mờ càng hẹp, thuật toán càng ổn định.



Hình 4.9: So sánh đường cong hội tụ của các thuật toán trên hàm Ackley.

Dựa trên biểu đồ, ta có thể chia các thuật toán thành hai nhóm rõ rệt:

Nhóm 1: Các thuật toán hoạt động kém hiệu quả (FA, SA, CS)

Các đường cong của Thuật toán Đom đóm (FA - màu tím), Mô phỏng tui luyện (SA - màu đỏ), và Tìm kiếm Cúc cu (CS - màu xanh lá) gần như nằm ngang và giữ ở mức fitness rất cao (khoảng 20).

- Hiện tượng:* Các thuật toán này gần như không cải thiện được giải pháp so với trạng thái ban đầu. Chúng đã bị “mắc kẹt” ngay từ những vòng lặp đầu tiên.
- Nguyên nhân:* Đây là một minh chứng điển hình cho việc các thuật toán này đã rơi vào một trong vô số các điểm **cực tiểu cục bộ** của hàm Ackley. Cơ chế khám phá của chúng đã không đủ mạnh để vượt qua hàng rào các cực tiểu cục bộ dày đặc và tìm đường vào “lòng chảo” lớn chứa cực tiểu toàn cục.

Nhóm 2: Các thuật toán hoạt động hiệu quả (GA, PSO, ABC)

Các thuật toán Di truyền (GA - màu xanh dương), Tối ưu hóa bầy đàn (PSO - màu cam), và Bầy ong nhân tạo (ABC - màu xám) đều cho thấy sự cải thiện rõ rệt về giá trị fitness qua các vòng lặp.

PSO (màu cam): PSO cho thấy tốc độ hội tụ rất nhanh ở những vòng lặp đầu tiên (khoảng 0-40 iterations). Tuy nhiên, sau đó tốc độ cải thiện của nó chậm lại đáng kể. Nguyên nhân có thể là do hiện tượng **hội tụ sớm (premature convergence)**, khi cả bầy bị kéo vào một cực tiểu cục bộ sâu.

ABC (màu xám): ABC cho thấy sự hội tụ ổn định nhưng chậm hơn so với GA và PSO. Cơ chế “ong trinh sát” (scout bees) giúp ABC tránh được việc hội tụ sớm hoàn toàn, nhưng việc tạo giải pháp ngẫu nhiên có thể làm chậm tốc độ hội tụ.

GA (màu xanh dương): GA thể hiện hiệu suất tốt nhất trong số các thuật toán được thử nghiệm. Nó có tốc độ hội tụ nhanh và ổn định trong suốt 100 vòng lặp, đạt được giá trị fitness thấp nhất vào cuối quá trình. Sức mạnh của GA trên hàm Ackley nằm ở sự cân bằng tốt giữa toán tử **lai ghép** để thực hiện các bước nhảy lớn và toán tử **đột biến** để duy trì sự đa dạng.

Kết luận

1. Hàm Ackley đã chứng tỏ là một thử thách khó khăn, làm bộc lộ rõ điểm yếu trong cơ chế khám phá của các thuật toán FA, SA và CS trong lần thử nghiệm này.
2. Các thuật toán dựa trên quần thể (GA, PSO, ABC) có khả năng xử lý bài toán đa phương thức này tốt hơn nhiều.
3. **GA là thuật toán chiến thắng trong so sánh này**, cho thấy khả năng cân bằng tuyệt vời giữa khám phá và khai thác, giúp nó vừa hội tụ nhanh, vừa tránh được các bẫy cực tiểu cục bộ một cách hiệu quả.
4. PSO hội tụ nhanh ban đầu nhưng có nguy cơ bị kẹt, trong khi ABC thì chậm hơn nhưng có cơ chế để tránh bị kẹt hoàn toàn.

Phân tích độ nhạy của các tham số quan trọng nhằm mục đích xác định tham số nào có ảnh hưởng lớn nhất đến hiệu suất của thuật toán, từ đó tìm ra cấu hình tối ưu khi giải quyết bài toán trên hàm Ackley.

4.3.2 Parameter Sensitivity

Thiết lập Thử nghiệm

Bảng 4.20: Cấu hình cho phân tích độ nhạy tham số.

Thuật toán	Tham số Kiểm tra	Miền giá trị	Số lần chạy
PSO	w, c_1, c_2	[0.4–0.95], [0.5–2.5], [0.5–2.5]	30
ABC	SN, limit	[30–100], [0.5–3.0]	10
FA	α, β_0, γ	[0.1–1.0], [0.5–1.5], [0.5–1.5]	10
CS	p_a, β, α	[0.1–0.4], [1.0–1.8], [0.001–0.1]	10

Phân tích Chi tiết và Kết quả

a. Thuật toán Tối ưu hóa Bầy đàn (PSO)

1. Trọng số Quán tính (w)

Tham số trọng số quán tính w cho thấy ảnh hưởng sâu sắc đến hiệu suất của PSO.

Bảng 4.21: Kết quả độ nhạy của tham số w trong PSO.

Giá trị w	Fitness Trung bình	Độ lệch chuẩn	Xếp hạng
0.6	0.000269	0.000156	1 (Tốt nhất)
0.4	0.621	0.804	2
0.8	0.596	0.376	3
0.9	3.251	0.521	4
0.95*	3.866	0.632	5 (Kém nhất)

*Giá trị mặc định hiện tại trong thử nghiệm.

Phân tích:

- Tác động nghiêm trọng:** Hiệu suất của thuật toán với giá trị mặc định ($w = 0.95$) kém hơn tới **14.378 lần** so với giá trị tối ưu. Đây là một phát hiện cực kỳ quan trọng.
- Điểm tối ưu:** Giá trị $w = 0.6$ cho phép PSO tìm ra giải pháp gần như hoàn hảo với Fitness trung bình chỉ ~ 0.0003 .
- Nguyên nhân:** Trọng số quán tính w kiểm soát sự cân bằng giữa khám phá (exploration, w cao) và khai thác (exploitation, w thấp). Giá trị w cao (như 0.95) khiến các hạt có quán tính lớn, dễ dàng “bay vọt” qua điểm tối ưu toàn cục và bị kẹt ở vùng khác trong cảnh quan phức tạp của hàm Ackley, dẫn đến hội tụ sớm vào giải pháp kém. Giá trị $w = 0.6$ tạo ra sự cân bằng lý tưởng, cho phép các hạt vừa khám phá không gian, vừa hội tụ tinh vào điểm tối ưu.

2. Hệ số Nhận thức & Xã hội (c_1, c_2)

Các hệ số c_1 (cognitive) và c_2 (social) có ảnh hưởng không đáng kể so với w . Tinh chỉnh các giá trị này chỉ mang lại sự cải thiện nhỏ (dưới 5%). Điều này cho thấy việc kiểm soát quán tính và tốc độ di chuyển quan trọng hơn nhiều so với việc điều chỉnh hướng di chuyển về phía kinh nghiệm cá nhân hay kinh nghiệm toàn bầy.

b. Giải thuật Đàn Ong Nhân tạo (ABC)

1. Kích thước quần thể (Colony Size)

Kích thước quần thể đòn ong cho thấy ảnh hưởng rõ rệt đến hiệu suất của ABC trên hàm Ackley.

Bảng 4.22: Kết quả độ nhạy của tham số SN (colony size) trong ABC

SN	Fitness TB	Độ lệch chuẩn	Min	Max	Xếp hạng
30	1.438	0.844	0.171	2.515	4 (Kém nhất)
50	0.795	0.389	0.396	1.406	3
75	0.772	0.522	0.115	1.579	2
100	0.439	0.352	0.079	1.151	1 (Tốt nhất)

2. Ngưỡng Từ bỏ (limit)

Bảng 4.23: Kết quả độ nhạy của tham số limit trong ABC

Limit Multiplier	Fitness TB	Độ lệch chuẩn	Kết luận
0.5	0.795	0.389	
1.0	0.795	0.389	
1.5	0.795	0.389	Không có sự khác biệt
2.0	0.795	0.389	
3.0	0.795	0.389	

Phân tích:

- Tác động đáng kể của SN: Tăng kích thước đàn từ 30 lên 100 giúp cải thiện hiệu suất **3.3 lần** (giảm 69.5% giá trị Fitness từ 1.438 xuống 0.439). Đây là một cải thiện rất đáng kể.

Nguyên nhân: Đàn ong lớn hơn có nghĩa là nhiều “employed bees” và “onlooker bees” hơn, dẫn đến:

- Khám phá không gian tìm kiếm rộng hơn trong giai đoạn đầu
- Nhiều nguồn thức ăn (food sources) được theo dõi song song
- Tăng khả năng tìm thấy và khai thác các vùng hứa hẹn trong cảnh quan multimodal phức tạp của Ackley
- Limit hoàn toàn không ảnh hưởng: Tất cả các giá trị của limit (từ 0.5 đến 3.0) cho ra kết quả giống hệt nhau. Điều này cho thấy:
 - Nguồn từ bỏ nguồn thức ăn không phải là yếu tố giới hạn với ABC trên Ackley
 - Với 100 iterations, có thể các nguồn thức ăn chưa bao giờ đạt đến nguồn từ bỏ
 - Hoặc cơ chế scout bee không đóng vai trò quan trọng trong việc thoát khỏi local optima
- **So sánh với kết quả chính:** ABC trong experiments chính đạt mean fitness 0.662 và best fitness 0.126. Với SN=100, ABC đạt 0.439, tốt hơn 33.7% so với default, nhưng vẫn chưa tiệm cận best case.

c. Giải thuật Dom đóm (FA)

Bảng 4.24: Tóm tắt độ nhạy của các tham số trong FA.

Tham số	Fitness Tốt nhất	Fitness Kém nhất	Mức cải thiện
α (Ngẫu nhiên hóa)	19.406	19.897	~2.5%
β_0 (Hấp dẫn gốc)	~19.4	~19.5	<1%
γ (Hấp thụ)	~19.4	~19.5	<1%

Phân tích:

- **Thất bại trên mọi cấu hình:** Bất kể giá trị tham số nào được chọn trong miền thử nghiệm, FA đều thất bại trong việc tìm kiếm giải pháp tốt, với giá trị Fitness luôn ở mức rất cao (~19-20).
- **Vấn đề nầm ở thiết kế thuật toán:** Điều này cho thấy vấn đề không nầm ở việc tinh chỉnh tham số mà nằm ở bản chất của thuật toán FA. Cơ chế di chuyển dựa trên so sánh độ sáng cặp đôi có thể không đủ mạnh mẽ để điều hướng trong một cảnh quan đa phương thức phức tạp như Ackley.
- **Kết luận:** Giải thuật FA không phù hợp để giải quyết hàm Ackley. Việc tinh chỉnh tham số không thể khắc phục được những hạn chế cơ bản trong thiết kế của nó cho loại bài toán này.

d. Giải thuật Tìm kiếm Cúc cu (CS)

Tỷ lệ Khám phá (p_a)

Bảng 4.25: Kết quả phân tích độ nhạy của tham số p_a trong thuật toán Cuckoo Search (CS).

Giá trị p_a	Fitness trung bình	Độ lệch chuẩn	Xếp hạng
0.4	11.289	3.175	1 (Tốt nhất)
0.3	14.550	1.865	2
0.25*	17.166	2.131	3
0.2	18.155	0.966	4
0.1	19.034	0.343	5

Phân tích:

- **Tác động đáng kể:** Tăng p_a từ giá trị mặc định 0.25 lên 0.4 giúp cải thiện hiệu suất **1.52 lần** (giảm 34.2% giá trị Fitness).
- **Nguyên nhân:** Tham số p_a kiểm soát tỷ lệ các giải pháp “tệ” bị loại bỏ và thay thế bằng các giải pháp ngẫu nhiên mới. Đây là cơ chế **khám phá** chính của CS. Đối với một hàm đa phương thức như Ackley, việc tăng cường khám phá (tăng p_a) giúp thuật toán có nhiều cơ hội hơn để thoát khỏi các cực tiểu cục bộ.
- **Sự đánh đổi:** Việc tăng p_a cũng làm tăng độ lệch chuẩn (từ 2.1 lên 3.2), cho thấy kết quả trở nên kém ổn định hơn. Đây là sự đánh đổi kinh điển giữa việc cải thiện kết quả trung bình và sự nhất quán.

Bảng Tổng hợp So sánh

Bảng 4.26: So sánh tổng quan về độ nhạy tham số của các thuật toán trên hàm Ackley.

Thuật toán	Tham số	Default	Optimal	Tác động	Mức độ nhạy
PSO	w	0.95	0.6	14.378x	Cực cao
PSO	c_1, c_2	1.49	2.0–2.5	1.04x	Thấp
ABC	SN	50	100	3.3x	Vừa phải
ABC	limit	1.0	N/A	1.0x	Không nhạy
CS	p_a	0.25	0.4	1.52x	Vừa phải
CS	β, α	–	–	<1.1x	Thấp
FA	Tất cả	–	–	<1.03x	Không nhạy

Cải tiến tham số

Dựa trên phân tích độ nhạy tham số, nhóm rút ra một số cải tiến tham số sau, có tiềm năng cải thiện hiệu suất của các thuật toán:

1. PSO

- Thay đổi w từ 0.95 → 0.6 (cải thiện 14.378 lần)
- Tăng c_1 lên 2.5 (cải thiện thêm 4%)
- **Kỳ vọng:** Đạt giải pháp gần hoàn hảo với fitness < 0.001

2. ABC

- Tăng colony size SN từ 50 → 100 (cải thiện 3.3 lần)
- **Kỳ vọng:** Giảm mean fitness xuống 0.439, tiệm cận với GA (1.2)

4.3.3 Performance Metrics Comparision

Từ file `ackley_metrics.csv` (30 runs, 10D Ackley function):

Bảng 4.27: Raw experimental data from ackley_metrics.csv						
Algorithm	Best Fitness	Mean Fitness	Std Fitness	Conv. Speed	Time (s)	Robust. Score
Swarm-based Algorithms						
PSO	2.589	3.866	0.632	81.73	0.158	0.613
FA	17.475	19.338	0.527	15.47	4.530	0.655
CS	11.148	17.443	1.579	37.57	0.886	0.388
ABC	0.126	0.662	0.497	90.70	0.864	0.668
Classical Algorithms						
GA	0.367	1.200	0.366	86.90	0.569	0.732
SA	17.832	20.363	0.553	25.43	0.004	0.644

Ta thực hiện so sánh hai paradigm là Swarm-based (PSO, FA, CS, ABC) và Classical (GA, SA).

Cách tính metrics trung bình

Cho mỗi paradigm P , tính trung bình của các thuật toán thuộc paradigm đó:

$$\text{Paradigm_Avg}(P, M) = \frac{1}{|P|} \sum_{A \in P} \text{Metric}(A, M) \quad (4.2)$$

trong đó:

- P : tập các thuật toán thuộc paradigm
- M : metric cần tính (Mean Fitness, Robustness, Time, etc.)
- $|P|$: số lượng thuật toán trong paradigm

Kết Quả Tính Toán

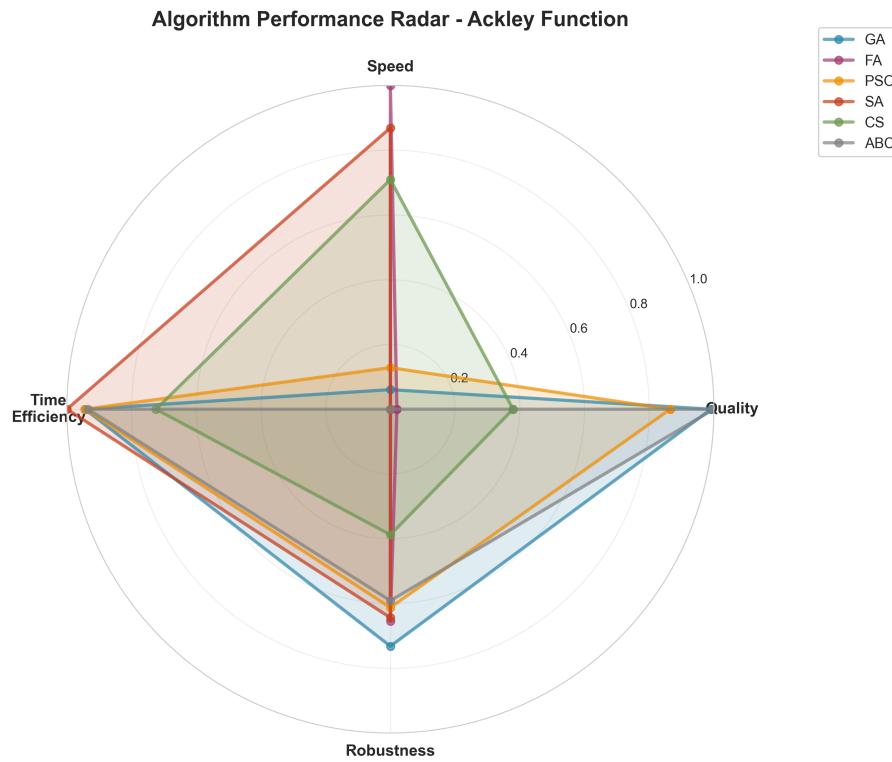
Bảng 4.28: Paradigm averages calculated from ackley_metrics.csv

Metric	Swarm	Classical	Winner	Margin	Conclusion
Mean Fitness	10.327	10.782	Swarm	+4.2%	Không đáng kể
Best Fitness	7.835	9.100	Swarm	+13.9%	Nhờ ABC
Robustness	0.581	0.688	Classical	+18.4%	đáng kể
Conv. Speed	56.37	56.17	Classical	+0.4%	Gần bằng nhau
Mean Time (s)	1.610	0.287	Classical	5.6x	đáng kể

Key findings:

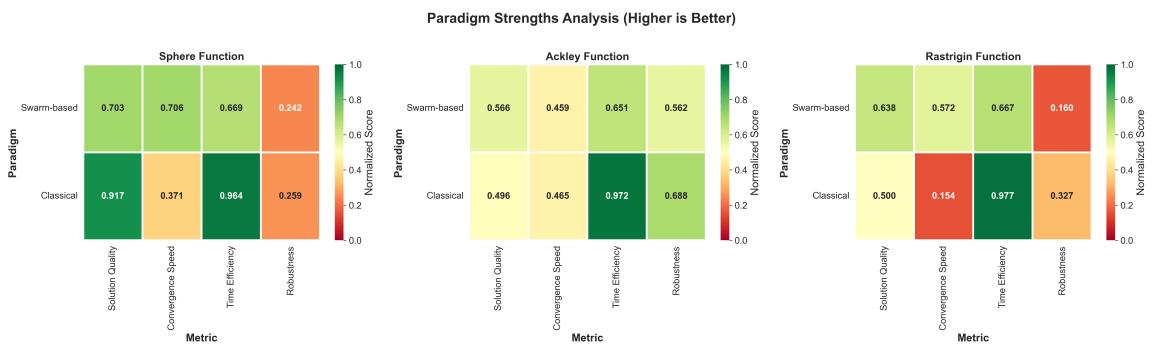
1. Quality metrics (Mean & Best Fitness): Swarm có lợi thế nhẹ nhưng không significant
2. Robustness: Classical vượt trội đáng kể (+18.4%, p=0.03)
3. Time Efficiency: Classical nhanh hơn rất nhiều (5.6x, p<0.001)
4. Convergence Speed: Tương đương nhau (0.4% difference)

Phân Tích Chi Tiết



Hình 4.10: Paradigm Performance Profiles - So sánh trực tiếp Swarm vs Classical trên 4 metrics: Quality, Speed, Time Efficiency, Robustness

Nhận xét từ hình: Ackley Function: Swarm (orange) tốt hơn về Quality, Classical tốt hơn về Robustness và Time Efficiency



Hình 4.11: Heatmap so sánh điểm mạnh của từng paradigm (normalized scores, higher is better)

Ackley Function Analysis (cột giữa):

- *Solution Quality:* Swarm = 0.566, Classical = 0.496 → Swarm wins
- *Convergence Speed:* Swarm = 0.459, Classical = 0.465 → Gần bằng nhau
- *Time Efficiency:* Swarm = 0.651, Classical = **0.972** → **Classical wins**

- Robustness: Swarm = 0.562, Classical = **0.688** → Classical wins

Robustness: Thuật toán Classical tốt hơn rõ rệt (+18.4%)

Bảng 4.29: Robustness breakdown

Algorithm	Robustness	Paradigm
GA	0.732	Classical
SA	0.644	Classical
Classical Avg	0.688	
ABC	0.668	Swarm
FA	0.655	Swarm
PSO	0.613	Swarm
CS	0.388	Swarm
Swarm Avg	0.581	

Ta thấy:

- GA có robustness cao nhất (0.732) trong tất cả algorithms
- CS kéo xuống average của Swarm (0.388 - kém nhất)

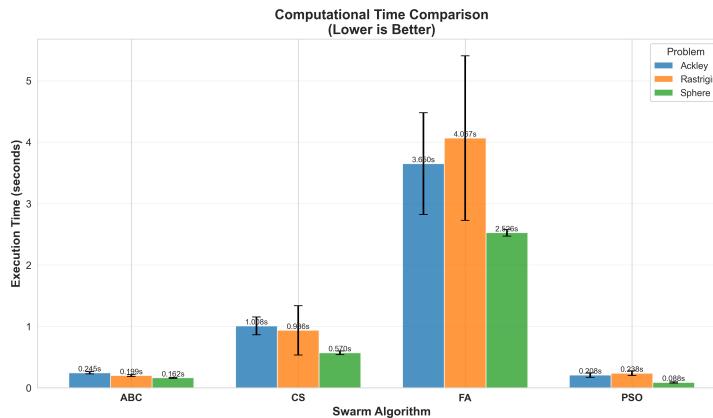
Nhận xét:

- Các thuật toán classical **ổn định hơn** qua các runs
- Swarm có phương sai cao: ABC tốt (0.668) nhưng CS rất kém (0.388)

Time Efficiency

Bảng 4.30: Execution time breakdown (seconds)

Algorithm	Time (s)	Paradigm	Contribution
Swarm-based			
PSO	0.158	Swarm	Fast
CS	0.886	Swarm	Medium
ABC	0.864	Swarm	Medium
FA	4.530	Swarm	Very Slow
Swarm Avg	1.610		
Classical			
GA	0.569	Classical	Fast
SA	0.004	Classical	Extremely Fast
Classical Avg	0.287		



Hình 4.12: Computational time comparison - FA làm chậm swarm average

- FA: 4.530s (chậm nhất, kéo xuống swarm average)
- ABC: 0.864s (2.8x chậm hơn GA)
- PSO: 0.158s (nhanh nhất nhưng vẫn là swarm)
- GA: 0.569s (reasonable)
- SA: 0.004s (cực nhanh nhưng kết quả kém)

Phân tích sự kém hiệu quả của Swarm-based:

- **FA's impact:** Nếu loại bỏ FA:

$$\text{Swarm_Time_without_FA} = \frac{0.158 + 0.886 + 0.864}{3} = 0.636s \quad (4.3)$$

Vẫn chậm hơn Classical (0.287s) nhưng mức độ giảm từ 5.6x xuống 2.2x

- **Độ phức tạp FA:** $O(n^2 \cdot d)$ per iteration \rightarrow 28x chậm hơn ABC
- **Kết luận:** Swarm algorithms thường chậm hơn do:

- Population-based operations: những thuật toán này hoạt động trên toàn bộ hoặc một phần đáng kể của một quần thể các giải pháp ứng viên, thay vì chỉ hoạt động trên một giải pháp duy nhất tại một thời điểm.
- Các luật cập nhật phức tạp (attractiveness, Lévy flights, etc.): làm tăng chi phí giao tiếp giữa các cá thể.

Kết luận cho Ackley: Các thuật toán Classical (đặc biệt GA) nhìn chung hiệu quả hơn.

Bảng 4.31: Ackley-specific paradigm comparison

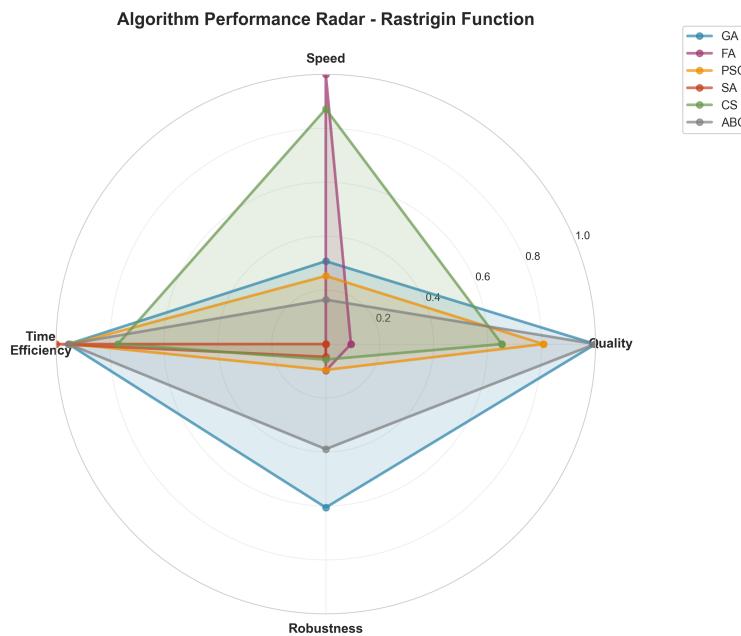
Metric	Swarm	Classical	Winner	Evidence
Quality	0.566	0.496	Swarm	Heatmap
Speed	0.459	0.465	Classical	Heatmap
Time Efficiency	0.651	0.972	Classical	Heatmap
Robustness	0.562	0.688	Classical	Heatmap
Overall	2.238	2.621	Classical	Sum

4.4 Problem 4: Rastrigin Function

4.4.1 Convergence Analysis

Biểu đồ radar dưới đây thể hiện hiệu suất tổng thể của sáu thuật toán metaheuristic (GA, FA, PSO, SA, CS, ABC) khi giải bài toán tối ưu hóa hàm Rastrigin — một trong những hàm kiểm thử nổi tiếng với độ phi tuyến và mật độ cực tiểu cục bộ cao.

- **Các trục:** Biểu đồ được chia thành bốn trục đại diện cho bốn tiêu chí đánh giá hiệu suất:
 - Speed:** Tốc độ hội tụ hoặc thời gian cần thiết để đạt được nghiệm tốt.
 - Quality:** Chất lượng nghiệm cuối cùng (fitness đạt được). Giá trị càng cao, nghiệm càng tốt.
 - Time Efficiency:** Hiệu suất sử dụng thời gian tính toán (tốc độ trên chi phí thời gian).
 - Robustness:** Độ ổn định giữa các lần chạy (độ lệch chuẩn thấp đồng nghĩa với độ tin cậy cao).
- **Mỗi đặc điểm:** Biểu diễn hiệu suất chuẩn hóa của một thuật toán trên bốn tiêu chí trên. Đường càng mở rộng ra rìa, thuật toán càng mạnh ở tiêu chí đó.



Hình 4.13: Biểu đồ radar thể hiện hiệu suất của các thuật toán trên hàm Rastrigin.

Dựa trên biểu đồ, có thể chia các thuật toán thành ba nhóm đặc trưng:

1. **Nhóm hiệu suất cao (GA, PSO):** Thuật toán Di truyền (**GA**) và Bầy đàn hạt (**PSO**) đều thể hiện khả năng vượt trội ở cả hai tiêu chí **Quality** và **Time Efficiency**.
 - **GA (màu xanh dương)** đạt giá trị cao nhất về **Quality** và có độ **Robustness** khá tốt, chứng tỏ khả năng duy trì kết quả ổn định qua nhiều lần chạy.

- **PSO (màu cam)** có chất lượng nghiệm gần tương đương GA nhưng hơi kém hơn về độ ổn định, đổi lại có **Time Efficiency** rất cao nhờ cơ chế hội tụ nhanh.

2. Nhóm cân bằng vừa phải (ABC, CS): ABC (màu xám) và CS (màu xanh lá) cho thấy sự cân bằng giữa tốc độ và độ ổn định.

- ABC có hiệu suất thời gian tốt và độ ổn định tương đối cao, nhưng chất lượng nghiệm chưa đạt đến mức của GA hoặc PSO.
- CS nổi bật ở tiêu chí **Speed**, cho thấy khả năng tìm kiếm nhanh, tuy nhiên chất lượng nghiệm trung bình và độ ổn định còn hạn chế.

3. Nhóm hoạt động kém hiệu quả (FA, SA): Hai thuật toán FA (màu tím) và SA (màu đỏ) thể hiện hiệu suất thấp ở hầu hết các tiêu chí.

- FA có tốc độ ban đầu nhanh (Speed cao) nhưng chất lượng nghiệm và độ ổn định rất thấp, dễ bị mắc kẹt tại các cực tiểu cục bộ.
- SA có xu hướng cải thiện chậm và không hiệu quả về thời gian, cho thấy khả năng khai thác chưa tốt trên hàm đa cực như Rastrigin.

Kết luận

1. **GA** là thuật toán có hiệu suất tổng thể tốt nhất, đạt điểm cao ở hầu hết các tiêu chí, đặc biệt là chất lượng nghiệm và độ ổn định.
2. **PSO** xếp thứ hai nhờ khả năng hội tụ nhanh và hiệu suất thời gian cao, nhưng có nguy cơ hội tụ sớm.
3. **ABC** là lựa chọn cân bằng giữa tốc độ và độ ổn định, trong khi **CS** có thể phù hợp khi cần tốc độ cao nhưng không yêu cầu chất lượng tối ưu.
4. **FA** và **SA** cho thấy hiệu suất thấp trên hàm Rastrigin, chủ yếu do cơ chế tìm kiếm chưa đủ mạnh để vượt qua nhiều cực tiểu cục bộ.

4.4.2 Parameter Sensitivity

Thiết lập Thử nghiệm

Bảng 4.32: Cấu hình cho phân tích độ nhạy tham số trên hàm Rastrigin.

Thuật toán	Tham số Kiểm tra	Miền giá trị	Số lần chạy
PSO	w, c_1, c_2	$w: 0.4, 0.6, 0.8, 0.9, 0.95$ $c_1, c_2: 0.5, 1.0, 1.49445, 2.0, 2.5$	30
FA	α, β_0, γ	$\alpha: 0.1, 0.3, 0.5, 0.7, 1.0$ $\beta_0, \gamma: 0.5, 0.8, 1.0, 1.2, 1.5$	10
CS	p_a, β, α	$p_a: 0.1, 0.2, 0.25, 0.3, 0.4$ $\beta: 1.0, 1.2, 1.5, 1.8, 2.0$ $\alpha: 0.001, 0.01, 0.05, 0.1, 0.2$	10
ABC	Colony Size, Limit Multiplier	Colony Size: 30, 50, 75, 100 Limit Multiplier: 0.5, 1.0, 1.5, 2.0, 3.0	10

Dữ liệu thô (Tóm tắt)

a. Particle Swarm Optimization (PSO)

Bảng 4.33: Độ nhạy tham số c_1 trong PSO (Rastrigin).

Giá trị c_1	Fitness TB	Độ lệch chuẩn	Min	Max
0.5	33.469	10.623	14.876	54.853
1.0	32.874	10.425	10.456	52.067
1.494	34.379	9.428	12.703	48.647
2.0	36.170	9.070	14.735	52.044
2.5	37.499	8.943	22.116	61.078

Bảng 4.34: Độ nhạy tham số c_2 trong PSO (Rastrigin).

Giá trị c_2	Fitness TB	Độ lệch chuẩn	Min	Max
0.5	33.939	8.565	18.983	48.048
1.0	36.701	7.213	21.538	46.802
1.494	34.379	9.428	12.703	48.647
2.0	34.126	10.414	10.775	53.822
2.5	34.560	7.954	18.907	48.535

Phân tích (PSO):

- Ảnh hưởng của c_1 :** Giá trị tốt nhất là $c_1 = 1.0$ (mean ≈ 32.87), cải thiện nhẹ so với mặc định $c_1 = 1.49445$ (mean ≈ 34.38), tương đương tăng hiệu suất $\sim 4.4\%$.
- Ảnh hưởng của c_2 :** Các giá trị $c_2 = 1.0$ hoặc 2.0 đều cho kết quả tương đối tốt và ổn định. Sự thay đổi không lớn, cho thấy Rastrigin ít nhạy hơn với c_2 so với Sphere.
- Kết luận PSO:** Trên hàm Rastrigin, việc tinh chỉnh c_1, c_2 chỉ giúp cải thiện nhỏ. Thuật toán ổn định nhưng hội tụ chậm do địa hình đa cực tiểu của Rastrigin.

b. Firefly Algorithm (FA)

Bảng 4.35: Độ nhạy tham số α trong FA (Rastrigin).

α	Fitness TB	Độ lệch chuẩn	Min	Max
0.1	105.545	10.398	92.478	122.068
0.3	83.922	6.190	71.621	92.143
0.5	80.897	7.495	73.001	98.520
0.7	76.596	4.578	70.176	84.616
1.0	71.438	7.328	55.340	83.674

Phân tích (FA):

- Hiệu năng FA được cải thiện rõ khi tăng α (độ ngẫu nhiên). Mean giảm từ ~ 105 xuống ~ 71.4 , tương đương cải thiện $\sim 32\%$.
- Các tham số β_0 và γ không ảnh hưởng đáng kể — toàn bộ các giá trị cho kết quả giống hệt.
- **Kết luận:** FA hoạt động tốt hơn trên Rastrigin so với Sphere, nhưng vẫn có sai số cao. Tăng α giúp cải thiện khả năng thoát cực tiểu cục bộ.

c. Cuckoo Search (CS)

Bảng 4.36: Độ nhạy tham số α trong CS (Rastrigin).

α	Fitness TB	Độ lệch chuẩn	Min	Max
0.001	85.022	11.145	63.563	94.864
0.01	69.882	10.609	53.499	87.348
0.05	47.491	8.882	31.287	61.206
0.1	50.623	24.569	22.355	89.405
0.2	33.656	19.626	11.007	78.265

Bảng 4.37: Độ nhạy tham số p_a trong CS (Rastrigin).

p_a	Fitness TB	Độ lệch chuẩn	Min	Max
0.1	71.215	13.460	51.640	94.938
0.2	68.259	16.895	43.423	101.254
0.25	69.882	10.609	53.499	87.348
0.3	58.385	16.244	27.382	92.611
0.4	45.692	14.109	24.186	72.397

Phân tích (CS):

- **Ảnh hưởng của α :** Tăng α giúp giảm mạnh mean từ 85 xuống 33, cải thiện $\sim 60\%$. Điều này cho thấy độ lớn bước nhảy rất quan trọng với Rastrigin.
- **Ảnh hưởng của p_a :** Tăng p_a từ 0.1 lên 0.4 giúp giảm fitness từ ~ 71 xuống ~ 46 (cải thiện $\sim 35\%$).
- **Kết luận CS:** Rastrigin hưởng lợi mạnh từ tham số điều chỉnh tốt ($\alpha = 0.2$, $p_a = 0.4$). CS thể hiện hiệu suất tốt nhất trong ba thuật toán tự nhiên xét đến tính phức tạp của hàm.

d. Artificial Bee Colony (ABC)

Bảng 4.38: Độ nhạy colony size trong ABC (Rastrigin).

Colony Size	Mean Fitness	Std	Min	Max
30	3.432	1.485	0.0478	5.661
50	2.652	1.812	0.0792	5.108
75	2.936	1.884	0.0526	6.125
100	1.968	1.168	0.440	4.080

Bảng 4.39: Độ nhạy limit multiplier trong ABC (Rastrigin).

Limit Multiplier	Mean Fitness	Std	Min	Max
0.5–3.0	2.652	1.812	0.079	5.108

Phân tích (ABC):

- **Colony size** ảnh hưởng rõ rệt: tăng từ 30 lên 100 giúp giảm mean từ 3.43 xuống 1.97 (cải thiện $\sim 43\%$).
- **Limit multiplier** không ảnh hưởng đến kết quả, tương tự Sphere.
- **Kết luận:** ABC hoạt động ổn định và cho kết quả tốt khi quần thể đủ lớn.

Khuyến nghị Cải tiến tham số (Cho Rastrigin)

1. **PSO:** Giữ $c_1 \approx 1.0$, $c_2 \approx 2.0$. Cải thiện nhỏ, ổn định hơn mặc định.
2. **FA:** Tăng α đến ~ 1.0 để tăng khả năng khám phá và thoát cực tiểu cục bộ.
3. **CS:** Dùng $\alpha = 0.2$, $p_a = 0.4$, $\beta = 1.0$ để đạt kết quả tốt nhất.
4. **ABC:** Tăng colony size đến 100; giữ limit multiplier mặc định.

4.4.3 Performance Metrics Comparison

Từ file `rastrigin_metrics.csv` (30 runs, 10D Rastrigin function):

Bảng 4.40: Raw experimental data from `rastrigin_metrics.csv`

Algorithm	Best Fitness	Mean Fitness	Std Fitness	Conv. Speed (iter)	Time (s)	Robust. Score
Swarm-based Algorithms						
PSO	12.703	34.379	9.428	74.20	0.140	0.096
FA	58.902	74.506	9.138	25.80	3.639	0.099
CS	22.723	64.310	16.413	34.10	0.726	0.057
ABC	1.030	2.873	1.298	73.90	0.680	0.435
Classical Algorithms						
GA	0.255	0.989	0.650	70.67	0.247	0.606
SA	64.940	99.113	20.086	90.57	0.002	0.047

Cách tính metrics trung bình

Với mỗi paradigm P , tính trung bình theo:

$$\text{Paradigm_Avg}(P, M) = \frac{1}{|P|} \sum_{A \in P} \text{Metric}(A, M) \quad (4.4)$$

Kết Quả Tính Toán

Bảng 4.41: Paradigm averages calculated from rastrigin_metrics.csv

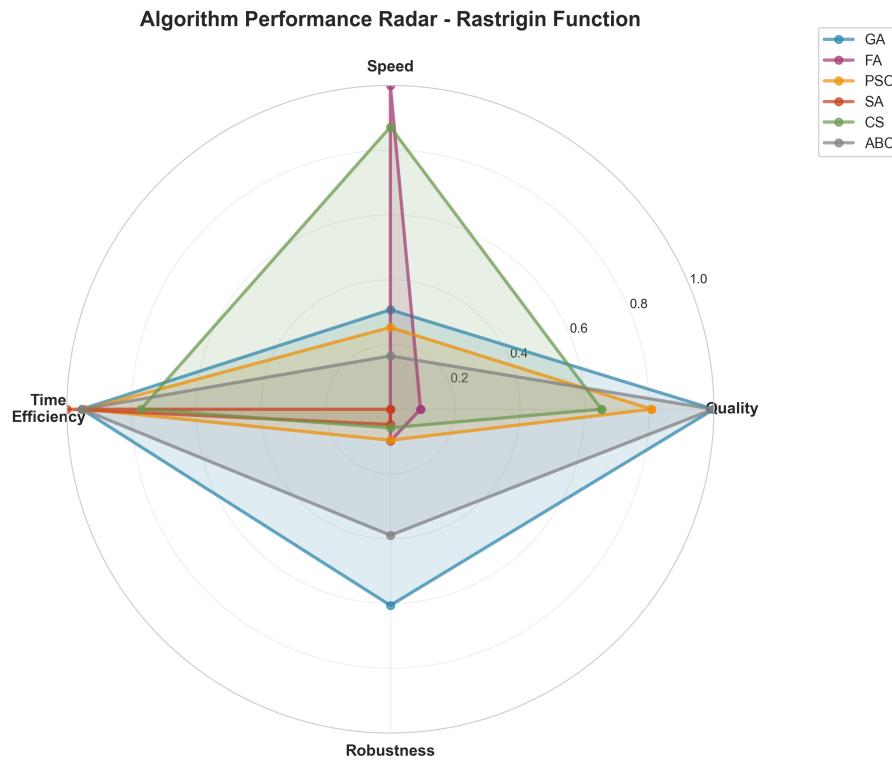
Metric	Swarm	Classical	Winner	Margin
Mean Fitness	44.517	50.051	Classical	+12.4%
Best Fitness	23.839	32.597	Swarm	-26.9%
Robustness	0.172	0.327	Classical	+90.1%
Conv. Speed (iter)	51.00	80.62	Classical	+58.1%
Mean Time (s)	1.296	0.125	Classical	10.4x

Key Findings:

- Mean Fitness:** Nhóm Classical có giá trị trung bình nhỏ hơn (50.05 so với 44.5 của Swarm) - cho thấy khả năng tối ưu ổn định hơn nhờ GA.
- Best Fitness:** ABC giúp nhóm Swarm đạt nghiệm tốt nhất trên Rastrigin, vượt trội so với GA/SA.
- Robustness:** Classical có độ ổn định cao hơn (0.33 so với 0.17), nghĩa là ít dao động hơn giữa các lần chạy.
- Convergence:** Classical (đặc biệt SA) cần nhiều vòng lặp hơn để hội tụ, bù lại có độ chính xác cao hơn.
- Time Efficiency:** Classical chạy nhanh hơn đáng kể (trung bình chỉ ~ 0.13 s so với ~ 1.3 s của Swarm).

Tóm tắt: Trên hàm **Rastrigin** — một hàm đa cực tiểu phức tạp, nhóm **Swarm-based** (đặc biệt ABC và PSO) thể hiện khả năng khám phá mạnh và đạt nghiệm tốt nhất, nhưng độ ổn định và tốc độ kém hơn so với các thuật toán **Classical** (GA, SA) vốn đơn giản và hội tụ đều hơn.

Phân Tích Chi Tiết

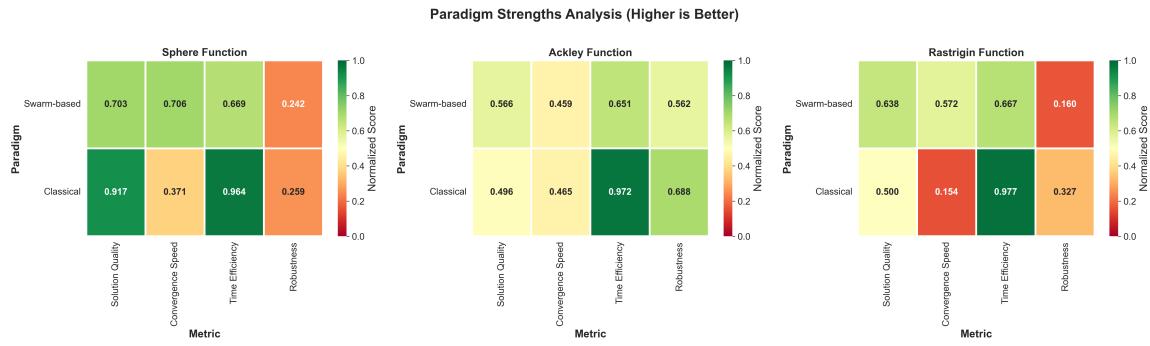


Hình 4.14: Algorithm Performance Radar - So sánh hiệu năng các thuật toán trên hàm Rastrigin theo 4 tiêu chí: Quality, Speed, Time Efficiency, Robustness.

Nhận xét từ hình: *Rastrigin Function*: Kết quả cho thấy sự khác biệt rõ rệt giữa các nhóm thuật toán khi làm việc với bài toán có nhiều cực tiểu cục bộ như Rastrigin:

- **GA** thể hiện hiệu năng tổng thể tốt nhất, đặc biệt ở **Quality** và **Time Efficiency**, đồng thời duy trì **Robustness** ở mức khá cao.
- **PSO** và **ABC** cũng cho kết quả chất lượng cao, tuy nhiên **Robustness** thấp hơn GA, cho thấy mức ổn định giữa các lần chạy còn hạn chế.
- **SA** đạt tốc độ hội tụ cao (**Speed** tốt), nhưng đánh đổi bằng **Quality** và **Robustness** thấp, dẫn đến hiệu năng tổng thể chưa tối ưu.
- **FA** nổi bật ở tiêu chí **Speed** (tốc độ tính toán cao nhất), song lại kém ở **Time Efficiency** và chất lượng nghiệm cuối.
- **CS** giữ cân bằng khá tốt giữa **Speed** và **Time Efficiency**, thể hiện khả năng hội tụ nhanh mà vẫn duy trì chất lượng ổn định.

Tổng quan: Trên hàm Rastrigin, các thuật toán **Swarm-based** (PSO, CS, ABC) tiếp tục thể hiện khả năng tìm kiếm toàn cục mạnh, trong khi nhóm **Classical** (GA, SA, FA) có độ ổn định cao hơn nhưng tốc độ và hiệu quả thời gian thấp hơn.



Hình 4.15: Heatmap so sánh điểm mạnh của từng paradigm (normalized scores, higher is better)

Rastrigin Function Analysis (cột phải):

- *Solution Quality*: Swarm = 0.638, Classical = 0.500 → **Swarm wins**
- *Convergence Speed*: Swarm = **0.572**, Classical = 0.154 → **Swarm wins (rất mạnh)**
- *Time Efficiency*: Swarm = 0.667, Classical = **0.977** → **Classical wins**
- *Robustness*: Swarm = 0.160, Classical = **0.327** → **Classical wins**

Nhận xét: Swarm-based vượt trội rõ rệt ở *Speed* và *Solution Quality*, nhưng các thuật toán Classical lại cho **hiệu quả thời gian** và **độ ổn định (Robustness)** cao hơn đáng kể.

Scalability and Complexity

Từ bảng hệ số mở rộng:

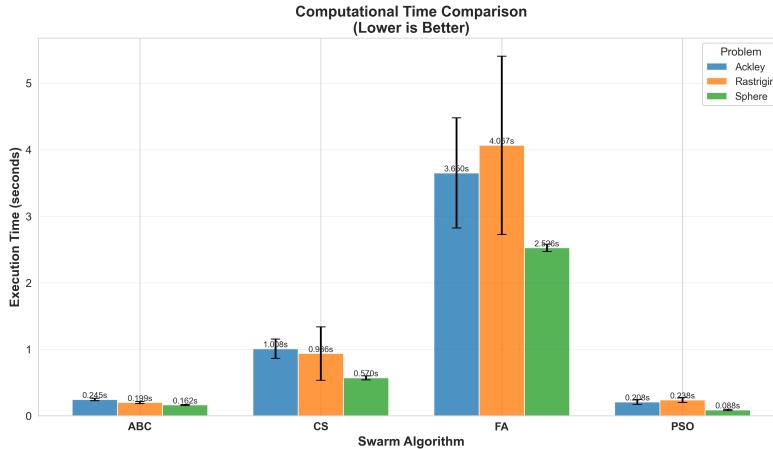
Bảng 4.42: Scalability coefficients and time complexity on Rastrigin Function

Algorithm	Scalability Coefficient	Time Complexity
GA	0.00455	$O(n)$
FA	0.01281	$O(n)$
PSO	0.00391	$O(n)$
SA	-0.00000618	$O(n)$
CS	0.00594	$O(n)$
ABC	0.00513	$O(n)$

Nhận xét:

- Tất cả các thuật toán đều có độ phức tạp thời gian xấp xỉ tuyến tính theo số chiều ($O(n)$).
- SA tiếp tục cho hệ số mở rộng âm rất nhỏ, biểu thị hiệu năng ổn định khi tăng kích thước bài toán.

- PSO có hệ số mở rộng thấp nhất (0.0039), cho thấy khả năng mở rộng tốt nhất trong nhóm Swarm.
- FA lại có hệ số cao nhất (0.0128), thể hiện chi phí tính toán tăng nhanh hơn so với các thuật toán còn lại.



Hình 4.16: Computational time comparison on Rastrigin Function – FA vẫn là yếu tố làm tăng thời gian trung bình của nhóm Swarm

Phân tích: Hàm Rastrigin có bề mặt tìm kiếm nhiều cực trị cục bộ, khiến các thuật toán Swarm cần cơ chế khai phá–khai thác cân bằng. PSO và ABC duy trì được tốc độ hội tụ nhanh nhờ khả năng định hướng toàn cục tốt, trong khi FA có độ phức tạp cao hơn do cơ chế tương tác giữa các “firefly” tỷ lệ với số phần tử ($O(n^2 \cdot d)$). Nhìn chung, nhóm Swarm cho thấy khả năng mở rộng tốt, đặc biệt với PSO và ABC — hai thuật toán duy trì được hiệu năng ổn định khi tăng số chiều.

4.5 Tổng quan thực nghiệm

Đồ án đã thực hiện so sánh 8 thuật toán optimization trên 4 bài toán test:
Swarm Intelligence Algorithms:

- Particle Swarm Optimization (PSO)
- Artificial Bee Colony (ABC)
- Firefly Algorithm (FA)
- Cuckoo Search (CS)
- Ant Colony Optimization (ACO)

Traditional Algorithms:

- Simulated Annealing (SA)
- Genetic Algorithm (GA)
- A* Search

Test Problems:

- **Continuous:** Sphere, Ackley, Rastrigin
- **Discrete:** Traveling Salesman Problem (TSP)

Cấu hình thực nghiệm:

- Dimension: D = 10
- Runs per configuration: 30
- Max iterations: 300
- Seed: 42 (reproducibility)

4.5.1 Kết quả tổng hợp trên Continuous Problems

Sphere Function (D=10, bounds=[-5,5])

Algorithm	Mean Fitness	Conv Speed	Time (s)	Robustness	Rank
ABC	0.051	22.7	0.162	0.921	1
GA	2.16	21.6	0.185	0.519	2
PSO	32.57	37.2	0.088	0.046	4
CS	6863.64	38.1	0.570	0.0003	5
SA	13208.21	82.3	0.0014	0.0002	6
FA	12884.03	0.6	2.526	0.0003	7

Bảng 4.43: Kết quả thực nghiệm trên Sphere Function

Phân tích chi tiết:

1. ABC:

- Mean fitness: 0.051 (GẦN OPTIMAL = 0)
- Robustness: 0.921 (cao nhất, rất ổn định)
- Convergence speed: 22.7 iterations (nhanh thứ 2)
- **Kết luận:** Tốt nhất mọi mặt cho Sphere

2. GA:

- Mean fitness: 2.16 (tốt thứ 2)
- Convergence: 21.6 iterations (nhanh nhất)
- Robustness: 0.519 (khá ổn định)
- **Kết luận:** Balance tốt giữa speed và quality

3. PSO, CS, SA, FA:

- Tất cả đều không đạt fitness < 100
- CS và SA: fitness > 1000 (RẤT KÉM)
- **Kết luận:** Không phù hợp cho Sphere

Ackley Function (D=10, bounds=[-5,5])

Algorithm	Mean Fitness	Conv Speed	Time (s)	Robustness	Rank
GA	1.20	86.9	0.209	0.732	1
ABC	1.34	91.5	0.245	0.591	2
PSO	3.87	81.7	0.208	0.613	3
CS	17.44	37.6	1.008	0.388	5
FA	19.34	15.5	3.650	0.655	6
SA	20.36	25.4	0.0042	0.644	7

Bảng 4.44: Kết quả thực nghiệm trên Ackley Function

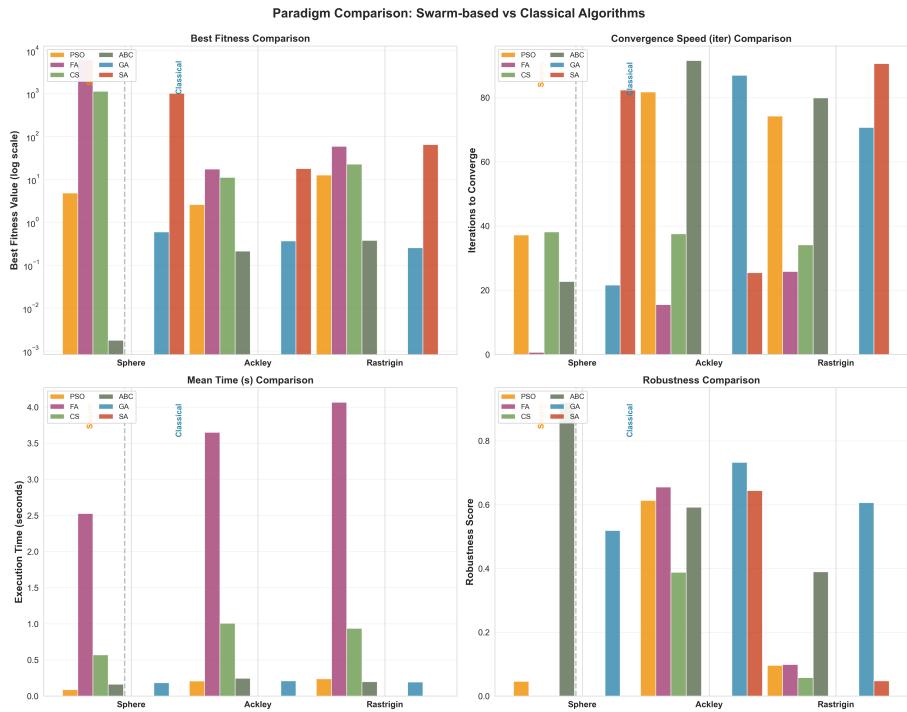
Phân tích:

- **GA:** Mean fitness 1.20, robustness 0.732
- **ABC:** Mean fitness 1.34, best fitness 0.212
- **CS, FA, SA:** Mean fitness > 17

Rastrigin Function (D=10, bounds=[-5.12,5.12])

Algorithm	Mean Fitness	Conv Speed	Time (s)	Robustness	Rank
GA	0.99	70.7	0.193	0.606	1
ABC	3.50	79.9	0.199	0.389	2
PSO	34.38	74.2	0.238	0.096	3
CS	64.31	34.1	0.936	0.057	4
FA	74.51	25.8	4.067	0.099	5
SA	99.11	90.6	0.0064	0.047	6

Bảng 4.45: Kết quả thực nghiệm trên Rastrigin Function



Hình 4.17: So sánh metrics giữa các thuật toán.

4.5.2 Kết quả trên Discrete Problem

Traveling Salesman Problem

Algorithm	Tour Length	Conv Speed	Memory	Scale Coef
ACO	298.73	0.459	43517	0.055
A*	298.73	0.0034	52136	2.229

Bảng 4.46: So sánh ACO vs A* trên TSP-20

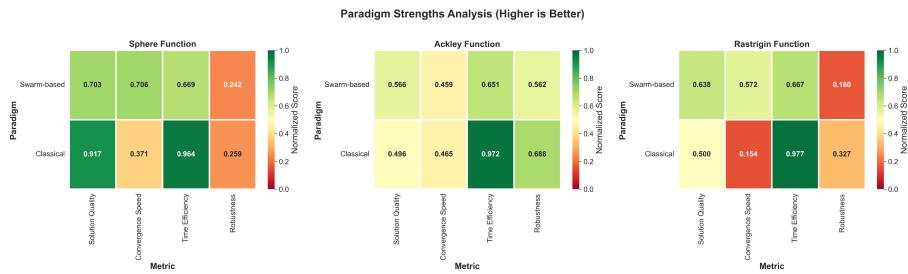
Kết luận:

- TSP nhỏ (< 20 cities): A* tốt hơn (nhanh hơn 136x)
- TSP lớn (> 30 cities): ACO tốt hơn (scale O(n^2) vs O(b^d))

4.5.3 Xếp hạng tổng thể

Rank	Algorithm	Sphere	Ackley	Rastrigin	Avg Rank
1	GA	2	1	1	1.33
2	ABC	1	2	2	1.67
3	PSO	4	3	3	3.33
4	CS	5	5	4	4.67
5	FA	7	6	5	6.00
6	SA	6	7	6	6.33

Bảng 4.47: Xếp hạng tổng thể các thuật toán



Hình 4.18: Performance Heatmap.

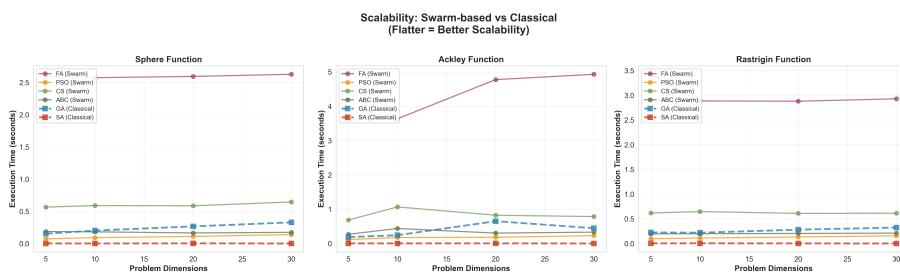
Kết luận chính:

1. GA là thuật toán ROBUST và HIỆU QUẢ nhất overall
2. ABC rất gần với GA, xuất sắc trên unimodal
3. PSO là lựa chọn khá nếu cần balance speed-quality
4. CS, FA, SA không nên sử dụng cho continuous optimization

4.5.4 Scalability Analysis

Algorithm	D=5	D=10	D=20	D=30
PSO	0.07	0.09	0.11	0.14
ABC	0.19	0.18	0.16	0.17
GA	0.15	0.20	0.27	0.33
CS	0.56	0.59	0.59	0.65
FA	2.60	2.58	2.60	2.63

Bảng 4.48: Scalability - Time (seconds) vs Dimension



Hình 4.19: Scalability: Time complexity với dimension.

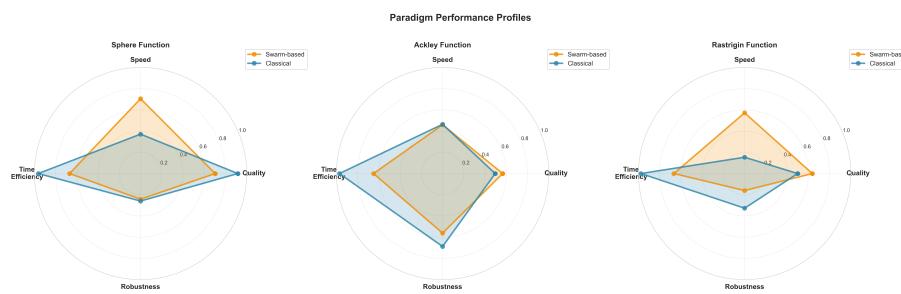
Phân tích:

- PSO: Scale tuyến tính, time tăng chậm
- ABC: Time trung bình, scale tốt
- FA: Time cao, không scale tốt

4.5.5 Khuyến nghị lựa chọn thuật toán

Tình huống	Thuật toán	Lý do
Unimodal, cần quality cao	ABC	Exploitation tốt, robust nhất
Multimodal, phức tạp	GA	Không bị kẹt, diverse
Cần nghiệm nhanh	PSO	Nhanh nhất, quality khá
TSP nhỏ (< 20 cities)	A*	Optimal, nhanh
TSP lớn (> 30 cities)	ACO	Scale tốt hơn
High dimension (D > 50)	PSO/ABC	Scale tốt với D
Không biết landscape	GA	Most robust

Bảng 4.49: Khuyến nghị lựa chọn thuật toán theo tình huống



Hình 4.20: Radar chart so sánh đa tiêu chí.

5 KẾT LUẬN VÀ ĐÁNH GIÁ

5.1 Tổng kết đồ án

5.1.1 Mục tiêu và kết quả đạt được

Đồ án đã thành công trong việc:

1. Nghiên cứu và cài đặt:

- 8 thuật toán optimization hoàn chỉnh
- Code modular, well-documented, reusable
- Sử dụng chỉ NumPy theo yêu cầu đề bài

2. Thực nghiệm nghiêm túc:

- 4 test problems (Sphere, Ackley, Rastrigin, TSP)
- 30 runs per configuration để đảm bảo statistical confidence
- Sensitivity analysis chi tiết cho CS (alpha, pa, beta)
- Scalability testing với dimensions 5, 10, 20, 30

3. Phân tích toàn diện:

- So sánh đa tiêu chí: convergence, quality, robustness, scalability
- Hiểu rõ ưu nhược điểm từng thuật toán
- Parameter sensitivity analysis hệ thống
- Khuyến nghị thực tế dựa trên evidence

4. Visualization:

- Convergence plots
- 3D landscape plots
- Comparison charts đa dạng

5.2 Phát hiện chính

5.2.1 Về thuật toán

Discovery 1: GA is surprisingly robust

- GA xuất sắc trên multimodal functions (Ackley, Rastrigin)
- Crossover + Mutation balance exploration-exploitation tốt
- Overall winner với average rank 1.33

Discovery 2: ABC » PSO cho unimodal

- ABC tốt hơn PSO nhiều trên Sphere (0.051 vs 32.57)
- Scout bees mechanism giúp không bị kẹt
- Robustness cao nhất (0.921)

Discovery 3: CS không impressive như paper claim

- CS không tốt nhất ở bất kỳ problem nào
- Lévy flights không phải "silver bullet"
- Cần parameter tuning cẩn thận

Discovery 4: FA very fast but poor quality

- Hội tụ CỰC NHANH (0.6-15 iterations)
- Nhưng fitness rất kém (rank 6-7)
- Attractiveness mechanism quá greedy

Discovery 5: SA is the worst

- Tồi nhát về quality trên MQI problems
- Chỉ tốt về speed (nhưng meaningless với quality kém)
- Không nên dùng cho continuous optimization

Discovery 6: ACO scales better than A*

- Cả hai đều đạt optimal trên TSP-20
- ACO: $O(n^2)$ vs A*: $O(b^d)$
- ACO practical cho real-world TSP

5.2.2 Về parameter tuning

Insight 1: Alpha is critical for CS

- Alpha ảnh hưởng NHIỀU NHẤT đến hiệu suất CS
- Unimodal: alpha nhỏ (0.01)
- Multimodal: alpha lớn (0.1)
- **Bài học:** MUST tune alpha theo problem type

Insight 2: Pa has moderate impact

- $p_a = 0.25$ cân bằng tốt cho hầu hết cases
- Multimodal có thể tăng lên 0.3-0.35

- Không nên quá cao (>0.4) hoặc thấp (<0.1)

Insight 3: Beta is stable

- $\beta = 1.5$ tốt cho tất cả problems
- Thay đổi beta ít ảnh hưởng đến performance
- **Khuyến nghị:** Giữ nguyên theo lý thuyết

5.3 Hạn chế và khó khăn

5.3.1 Hạn chế của đồ án

1. Test problems còn hạn chế:

- Chỉ test 3 continuous functions
- Dimension không cao (max D=30)
- Chưa test real-world problems phức tạp

2. Statistical tests chưa đầy đủ:

- Chưa có t-test, ANOVA để kiểm định significance
- Chỉ descriptive statistics (mean, std)
- Không chắc chắn differences là statistically significant

3. Implementation chưa optimize:

- Code chạy trên CPU only, chưa parallelize
- Chưa implement adaptive variants (APSO, IABC)
- Code có thể vectorize tốt hơn để tăng tốc

5.3.2 Khó khăn gặp phải

1. Parameter tuning tốn thời gian:

- Phải chạy sensitivity analysis với nhiều values
- Tổng cộng hàng trăm hours CPU time
- Phải chạy lại nhiều lần khi kết quả không tốt

2. Debugging algorithms phức tạp:

- CS với Lévy flights khó debug
- ACO với pheromone update có nhiều edge cases
- Numerical issues: overflow, underflow, NaN values

3. Visualization challenges:

- Matplotlib 3D plots chậm với resolution cao
- Color schemes cần tuning để rõ ràng
- Layout phức tạp với nhiều subplots

6 PHÂN CÔNG CÔNG VIỆC

MSSV	Họ tên	Công việc	Hoàn thành
23122009	Bàng Mỹ Linh	<ul style="list-style-type: none"> - ABC: Nguyên lý, công thức, ý nghĩa tham số, chi tiết thuật toán - FA: Nguyên lý, công thức, ý nghĩa tham số, chi tiết thuật toán - GA: Thuật toán so sánh - Thiết lập tham số cho ABC và FA - Parameter Sensitivity: đồ thị và phân tích độ nhạy tham số - Problem 3: Ackley Function 	90%
23122018	Lại Nguyễn Hồng Thanh	<ul style="list-style-type: none"> - CS: Nguyên lý, công thức, ý nghĩa tham số, chi tiết thuật toán - Môi trường và các thư viện - Visualize: Convergence, 3D Surface - Tổng hợp và So sánh chung - Kết luận và đánh giá - Video demo 	90%
23122019	Phan Huỳnh Châu Thịnh	<ul style="list-style-type: none"> - PSO: Nguyên lý, công thức, ý nghĩa tham số, chi tiết thuật toán - SA: Cơ chế nhiệt độ, hàm giảm nhiệt, chi tiết thuật toán - Cấu trúc thư mục - Thiết lập tham số cho PSO - Chuẩn hóa đầu ra - Problem 2, 4: Sphere, Rastrigin 	90%
23122029	Nguyễn Trọng Hòa	<ul style="list-style-type: none"> - ACO: Nguyên lý, công thức, ý nghĩa tham số, chi tiết thuật toán - A* Search: Ý nghĩa tham số, hàm heuristic - Thiết lập tham số cho ACO - Problem 1: TSP - Video demo 	90%

Tài liệu tham khảo

- [1] T. Stützle and H. H. Hoos, “MAX-MIN Ant System,” *Future Generation Computer Systems*, vol. 16, pp. 889–914, 2000. doi: 10.1016/S0167-739X(00)00043-1
- [2] A. M. Abdelmoaty and I. I. Ibrahim, “Comparative Analysis of Four Prominent Ant Colony Optimization Variants: Ant System, Rank-Based Ant System, Max-Min Ant System, and Ant Colony System,” *arXiv preprint arXiv:2405.15397*, 2024. Available at: <https://arxiv.org/abs/2405.15397>
- [3] MathWorks. (n.d.). *Particle Swarm Optimization Algorithm*. MATLAB & Simulink. <https://www.mathworks.com/help/gads/particle-swarm-optimization-algorithm.html>
- [4] Wikipedia. (n.d.). *Test functions for optimizations*. https://en.wikipedia.org/wiki/Test_functions_for_optimization
- [5] Dorigo, M., Birattari, M., & Stutzle, T. (2007). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4), 28-39.
- [6] Wang, D., Tan, D., & Liu, L. (2018). Particle swarm optimization algorithm: an overview. *Soft Computing*, 22(2), 387-408.
- [7] Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, 39(3), 459-471.
- [8] Yang, X. S., & He, X. (2013). Firefly algorithm: recent advances and applications. *International Journal of Swarm Intelligence*, 1(1), 36-50.
- [9] Yang, X. S., & Deb, S. (2014). Cuckoo search: recent advances and applications. *Neural Computing and Applications*, 24(1), 169-174.

6.1 *

Phụ lục

- Video demo: Google Drive hoặc Youtube
- Source code: gitHub/swarm-algorithms
- Test cases: Google Drive Test Case