

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC NGUYỄN TẤT THÀNH
KHOA CÔNG NGHỆ THÔNG TIN**



TIỂU LUẬN MÔN TỐI ƯU HÓA TỔ HỢP

**Tên đề tài: ỨNG DỤNG THUẬT
TOÁN ACO VÀO TÌM KIẾM CÂY
XÃNG GẦN NHẤT**

Giảng viên hướng dẫn: TS. CAO VĂN KIÊN

Học viên thực hiện:

NGUYỄN NGỌC PHƯƠNG QUỲNH MSSV: 2400000006

HOÀNG NGUYỄN MINH TUẤN MSSV: 2400000053

Khoá: 2024 – 2025

Ngành/ chuyên ngành: CÔNG NGHỆ THÔNG TIN

Tp HCM, tháng 09 năm 2024

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC NGUYỄN TẤT THÀNH
KHOA CÔNG NGHỆ THÔNG TIN**



TIỂU LUẬN MÔN TỐI ƯU HÓA TỔ HỢP

**Tên đề tài: ỨNG DỤNG THUẬT
TOÁN ACO VÀO TÌM KIẾM CÂY
XÃNG GẦN NHẤT**

Giảng viên hướng dẫn: TS. CAO VĂN KIÊN

Học viên thực hiện:

NGUYỄN NGỌC PHƯƠNG QUỲNH MSSV: 2400000006

HOÀNG NGUYỄN MINH TUẤN MSSV: 2400000053

Khoá: 2024 – 2025

Ngành/ chuyên ngành: CÔNG NGHỆ THÔNG TIN

Tp HCM, tháng 09 năm 2024

LỜI CẢM ƠN

Trước tiên, em xin gửi lời cảm ơn chân thành đến Thầy Cao Văn Kiên – người đã nhiệt tình hướng dẫn, hỗ trợ và đóng góp ý kiến quý báu trong suốt quá trình thực hiện tiểu luận này. Những lời khuyên và sự chỉ dẫn tận tâm của Thầy đã giúp em hoàn thành tiểu luận một cách tốt nhất.

Em cũng xin chân thành cảm ơn các bạn cùng lớp và đồng nghiệp đã luôn đồng hành, động viên và hỗ trợ cho em thêm nhiều hướng giải quyết và khuyết mắc cũng như những khó khăn trong quá trình nghiên cứu và viết tiểu luận. Tuy nhiên do thời gian tương đối ngắn để có thể hoàn thiện bài nghiên cứu tốt nhất nên sẽ còn nhiều sai sót trong quá trình làm bài, mong cô thông cảm và góp ý thêm cho bài của chúng em.

Cuối cùng, em xin cảm ơn tất cả những ai đã trực tiếp hay gián tiếp giúp đỡ em trong quá trình hoàn thành tiểu luận này. Những ý kiến đóng góp và sự giúp đỡ của mọi người là nguồn động lực quý giá giúp em hoàn thành tốt công việc.

Em xin chân thành cảm ơn!

LỜI MỞ ĐẦU

Trong thời đại số hóa và bùng nổ dữ liệu, việc xử lý và tối ưu hóa các hệ thống thông tin trở thành một trong những thách thức lớn đối với cả các nhà nghiên cứu và các chuyên gia trong ngành. Đặc biệt các bài toán có liên quan đến tìm kiếm và phân tích dữ liệu là một trong các vấn đề được quan tâm nhiều nhất hiện nay, do tính ứng dụng cao vào thực tiễn như định vị, tối ưu hóa hành trình trong logistics, hệ thống network, ... Với khối lượng dữ liệu khổng lồ theo thời gian, nhu cầu về phương pháp giải quyết bài toán cũng sẽ có những đòi hỏi tốt hơn cho mỗi một hướng giải quyết được đưa ra. Phương pháp đó cần đáp ứng được mức độ hiệu quả, chính xác và tiết kiệm tài nguyên.

Thuật toán đàn kiến (Ant Colony Optimization - ACO) là một phương pháp tối ưu hóa sử dụng hình thức mô phỏng tự nhiên. Thuật toán ra đời dựa vào việc mô phỏng theo hành vi tìm kiếm thức ăn của loài kiến. Trong tự nhiên, theo sự hình thành và phát triển, đàn kiến đã tự tạo ra tập tính tiếp cận nguồn thức ăn tốt nhất cũng như đánh dấu lại con đường đó bằng cách tiết ra pheromone để truyền tín hiệu cho thành viên trong đàn.

Việc ứng dụng ACO vào bài toán tìm kiếm gần nhất là một cách tạo ra hướng tiếp cận đơn giản dễ áp dụng hơn. Vì thuật toán này không đòi hỏi xây dựng một mô hình quá phức tạp, thay vào đó sẽ sử dụng dựa trên quyết định cục bộ và thông tin phân tán. Thuật toán ACO mang lại nhiều lợi ích về chi phí tính toán, khả năng ứng dụng cao để tìm ra giải pháp tối ưu trong một tập dữ liệu lớn, đặc biệt là trong các vấn đề liên quan đến lộ trình và định tuyến.

Tiểu luận này sẽ tập trung nghiên cứu việc ứng dụng thuật toán đàn kiến vào bài toán tìm kiếm gần nhất, bắt đầu từ việc trình bày các lý thuyết cơ bản, tổng quan tài liệu liên quan, đến việc phát triển và áp dụng thuật toán trong môi trường thực nghiệm. Bằng cách phân tích các kết quả thu được, chúng tôi hy vọng sẽ làm sáng tỏ được hiệu quả của ACO trong việc giải quyết bài toán này, đồng thời đề

xuất các hướng nghiên cứu tiếp theo nhằm cải thiện và mở rộng ứng dụng của thuật toán trong các lĩnh vực khác.

Trong suốt quá trình thực hiện tiểu luận, chúng tôi đã nhận được sự hỗ trợ to lớn từ các giảng viên và bạn bè. Những ý kiến đóng góp, sự hướng dẫn tận tình và sự động viên từ họ là nguồn động lực lớn giúp chúng tôi hoàn thành nghiên cứu này. Chúng tôi xin bày tỏ lòng biết ơn sâu sắc đến tất cả những người đã đồng hành cùng chúng tôi trong hành trình này.

TRƯỜNG ĐẠI HỌC NGUYỄN TẤT THÀNH
TRUNG TÂM KHẢO THÍ

KỲ THI KẾT THÚC HỌC PHẦN
HỌC KỲ HK1 NĂM HỌC 2024 - 2025

PHIẾU CHẤM THI TIỂU LUẬN/ĐỒ ÁN

Môn thi: Tối ưu hóa tổ hợp

Lớp học phần: 1101073315

Nhóm sinh viên thực hiện:

1. Nguyễn Ngọc Phương Quỳnh..... Tham gia đóng góp: tìm hiểu tài liệu, trình bày, viết tài liệu

2. Hoàng Nguyễn Minh Tuấn..... Tham gia đóng góp: code, xử lý thuật toán, tìm hiểu tài liệu, thiết kế ứng dụng

3. Tham gia đóng góp:.....

4. Tham gia đóng góp:.....

5. Tham gia đóng góp:.....

Ngày thi: 06/09/2024..... Phòng thi: D.206

Đề tài tiểu luận/báo cáo của học viên: Ứng dụng thuật toán aco vào tìm kiếm cây xăng gần nhất.

Phân đánh giá của giảng viên (căn cứ trên thang rubrics của môn học):

Tiêu chí (theo CDR HP)	Đánh giá của GV	Điểm tối đa	Điểm đạt được
Cấu trúc của báo cáo		
Nội dung			
- Các nội dung thành phần		
- Lập luận		
- Kết luận		
Trình bày		
TỔNG ĐIỂM			

Giảng viên chấm thi
(ký, ghi rõ họ tên)

MỤC LỤC

LỜI CẢM ƠN	i
LỜI MỞ ĐẦU	ii
MỤC LỤC	v
DANH MỤC HÌNH	vii
CHƯƠNG 1 GIỚI THIỆU CHUNG.....	1
1.1 Mục đích và ý nghĩa của nghiên cứu	1
1.2 Mục tiêu của nghiên cứu.....	1
1.3 Phương pháp nghiên cứu.....	2
1.4 Cấu trúc của tiểu luận	2
CHƯƠNG 2 : TỔNG QUAN TÀI LIỆU VÀ CƠ SỞ LÝ THUYẾT	5
2.1 Tổng quan về tối ưu tổ hợp	5
2.2 Tổng quan về bài toán tìm kiếm gần nhất	7
2.3 Giới thiệu về thuật toán đàn kiến (ACO).....	7
2.4 Liên hệ giữa ACO và bài toán tìm kiếm gần nhất	8
2.5 Ứng dụng ACO trong tìm kiếm gần nhất	8
2.6 Cách tiếp cận	9
2.6.1 Heuristic cấu trúc.....	9
2.6.2 Tìm kiếm cục bộ (local search).	10
2.6.3 Metaheuristic.....	11
2.6.4 Công cụ và phương pháp.....	12
2.6.5 Thiết kế thí nghiệm và quy trình thực hiện nghiên cứu	12
2.6.6 Thuật toán Rank-Based Ant System (RBAS).....	13
2.6.7 Các nghiên cứu liên quan	13
CHƯƠNG 3 : PHƯƠNG PHÁP THỰC HIỆN.....	15
3.1 Mô tả bài toán	15
3.2 Ứng dụng ACO trong tìm kiếm gần nhất.....	15
3.3 Ứng dụng trong tìm kiếm điểm gần nhất	17
3.4 Giải pháp:	18
3.5 Tổng kết:.....	30
CHƯƠNG 4 : KẾT QUẢ VÀ THẢO LUẬN	32

4.1	Trình bày kết quả:	32
4.2	Phân tích kết quả:	33
	CHƯƠNG 5 : KẾT LUẬN VÀ ĐỀ NGHỊ.....	34
5.1	Kết luận	34
5.2	Đề nghị.....	36
	PHỤ LỤC	1
	DANH MỤC TÀI LIỆU THAM KHẢO	37

DANH MỤC HÌNH

Hình 2.1 Phương pháp heuristic cấu trúc tham ăn	10
Hình 2.2 Lời giải nhận được nhờ thay 2 cạnh (2,3), (1,6) bằng (1,3), (2,6)	11
Hình 2.3 Thuật toán memetic sử dụng EC	12
Hình 3.1 Công thức tính xác suất lựa chọn	16
Hình 3.2 Công thức tính lượng pheromone	17
Hình 3.3 Thuật toán NNS.....	19
Hình 3.4 Kết quả các điểm gần nhất	19
Hình 3.5 Thuật toán ACO	20
Hình 3.6 Kết quả các điểm gần nhất	21
Hình 3.7 Biểu đồ so sánh chi phí thời gian giữa ACO và NNS.....	22
Hình 3.8 Công thức tính xác suất lựa chọn P_{ij}	23
Hình 3.9 Biểu đồ so sánh chi phí thời gian giữa ACO và ACS.....	24
Hình 3.10 Biểu đồ so sánh chi phí thời gian ACO và MMAS	25
Hình 3.11 Biểu đồ so sánh chi phí thời gian ACO và EAS	27
Hình 3.12 Biểu đồ so sánh chi phí thời gian ACO và RAS	28
Hình 3.13 Biểu đồ so sánh chi phí thời gian ACO và GA+ACO	29
Hình 4.1 Biểu đồ so sánh chi phí thời gian ACO và GA + ACO	32
Hình 4.2 Biểu đồ so sánh chất lượng giải pháp ACO và GA + ACO.....	33

CHƯƠNG 1 GIỚI THIỆU CHUNG

1.1 Mục đích và ý nghĩa của nghiên cứu

Trong bối cảnh sự phát triển nhanh chóng của khoa học và công nghệ, các bài toán tìm kiếm và tối ưu hóa ngày càng đóng vai trò quan trọng trong việc xử lý và phân tích dữ liệu lớn. Bài toán tìm kiếm gần nhất, một bài toán cơ bản trong khoa học máy tính và trí tuệ nhân tạo, liên quan đến việc tìm kiếm đối tượng hoặc điểm gần nhất trong một tập hợp lớn các đối tượng khác. Nó có ứng dụng rộng rãi trong nhiều lĩnh vực như định vị, phân loại, và hệ thống thông tin địa lý.

Thuật toán đàn kiến (Ant Colony Optimization - ACO) đã được chứng minh là một công cụ mạnh mẽ trong việc giải quyết các bài toán tối ưu hóa phức tạp nhờ vào khả năng mô phỏng hành vi tìm kiếm thức ăn của loài kiến. Việc áp dụng ACO vào bài toán tìm kiếm gần nhất không chỉ hứa hẹn mang lại những cải tiến về hiệu quả tính toán mà còn mở ra những hướng đi mới trong việc phát triển các thuật toán tối ưu hóa.

Nghiên cứu này nhằm mục đích khám phá và đánh giá hiệu quả của thuật toán ACO trong việc giải quyết bài toán tìm kiếm gần nhất, đồng thời xác định các yếu tố ảnh hưởng đến hiệu suất của thuật toán và đề xuất các cải tiến có thể áp dụng.

1.2 Mục tiêu của nghiên cứu

Nghiên cứu này hướng đến việc xây dựng một hệ thống tối ưu hóa dựa trên thuật toán ACO (Ant Colony Optimization) để tìm kiếm cây xăng gần nhất trong các khu vực đô thị. Mục tiêu tổng quát là ứng dụng một thuật toán metaheuristic mạnh mẽ nhằm giải quyết bài toán tối ưu hóa trong lĩnh vực định vị không gian, với khả năng mở rộng ra các ứng dụng tìm kiếm khác trong tương lai.

Để cụ thể hóa mục tiêu này, nghiên cứu sẽ tập trung vào việc áp dụng thuật toán ACO để mô phỏng quá trình tìm kiếm cây xăng gần nhất từ vị trí hiện tại của người dùng. Thông qua mô phỏng này, chúng tôi sẽ đánh giá hiệu quả của thuật toán ACO trong việc xác định đường đi tối ưu, dựa trên các yếu tố như khoảng cách, thời gian di chuyển, và điều kiện giao thông. Bên cạnh đó, nghiên cứu cũng đặt ra mục tiêu so sánh hiệu suất của thuật toán ACO với các thuật toán tìm kiếm

truyền thống, chẳng hạn như Dijkstra và BFS, nhằm xác định những lợi thế và hạn chế của ACO trong bối cảnh này. Cuối cùng, nghiên cứu sẽ phân tích và đánh giá các yếu tố ảnh hưởng đến hiệu quả của thuật toán ACO, bao gồm số lượng cây xăng, cấu trúc địa hình, và thời gian xử lý, để cung cấp cái nhìn toàn diện về khả năng ứng dụng của ACO trong thực tế.

1.3 Phương pháp nghiên cứu

Phương pháp tiếp cận của nghiên cứu sẽ bắt đầu với việc tìm hiểu lý thuyết và tổng quan tài liệu liên quan đến thuật toán ACO và các ứng dụng thực tế của nó trong lĩnh vực tối ưu hóa. Đồng thời, chúng tôi sẽ khảo sát các phương pháp tìm kiếm đường đi hiện có, bao gồm các thuật toán truyền thống như Dijkstra và BFS, để làm cơ sở so sánh với ACO.

Sau đó, nghiên cứu sẽ tiến hành thiết kế một mô hình mô phỏng sử dụng thuật toán ACO, trong đó các vị trí cây xăng được biểu diễn dưới dạng các nút trong một mạng lưới đồ thị. Mô hình này sẽ mô phỏng hành vi của các con kiến trong quá trình tìm kiếm đường đi tối ưu đến cây xăng gần nhất, với các tham số như khoảng cách và thời gian được đưa vào.

Về mặt thực nghiệm, chúng tôi sẽ tiến hành các thử nghiệm với dữ liệu giả lập về vị trí của cây xăng và các điểm xuất phát khác nhau. Dữ liệu thu được từ các thử nghiệm này sẽ được phân tích để đánh giá hiệu quả của thuật toán ACO, tập trung vào các chỉ số như thời gian xử lý và độ chính xác trong việc tìm kiếm cây xăng gần nhất. Cuối cùng, kết quả từ thuật toán ACO sẽ được so sánh với kết quả từ các thuật toán truyền thống như Dijkstra và BFS, nhằm đánh giá các chỉ số hiệu suất như thời gian xử lý, khoảng cách đến cây xăng gần nhất, và khả năng thích ứng với các điều kiện giao thông khác nhau. Qua đó, nghiên cứu sẽ đưa ra kết luận về tính khả thi và hiệu quả của việc ứng dụng thuật toán ACO trong việc tìm kiếm cây xăng gần nhất.

1.4 Cấu trúc của tiểu luận

Tiểu luận bao gồm 05 chương:

CHƯƠNG 1 GIỚI THIỆU CHUNG

1.1 Mục đích và ý nghĩa của nghiên cứu

1.2 Mục tiêu của nghiên cứu

1.3 Phương pháp nghiên cứu

1.4 Cấu trúc của tiểu luận

CHƯƠNG 2 : TỔNG QUAN TÀI LIỆU VÀ CƠ SỞ LÝ THUYẾT

2.1 Tổng quan về tối ưu tổ hợp

2.2 Tổng quan về bài toán tìm kiếm gần nhất

2.3 Giới thiệu về thuật toán đàn kiến (ACO)

2.4 Liên hệ giữa ACO và bài toán tìm kiếm gần nhất

2.5 Ứng dụng ACO trong tìm kiếm gần nhất

2.6 Cách tiếp cận

2.6.1 Heuristic cấu trúc

2.6.2 Tìm kiếm cục bộ (local search)

2.6.3 Metaheuristic

2.6.4 Công cụ và phương pháp

2.6.5 Thiết kế thí nghiệm và quy trình thực hiện nghiên cứu

2.6.6 Thuật toán Rank-Based Ant System (RBAS)

2.6.7 Các nghiên cứu liên quan

CHƯƠNG 3 : PHƯƠNG PHÁP THỰC HIỆN

3.1 Mô tả bài toán

3.2 Ứng dụng ACO trong tìm kiếm gần nhất

3.3 Ứng dụng trong tìm kiếm điểm gần nhất

3.4 Giải pháp

3.5 Tổng kết

CHƯƠNG 4 : KẾT QUẢ VÀ THẢO LUẬN

4.1 Trình bày kết quả

4.2 Phân tích kết quả

CHƯƠNG 5 : KẾT LUẬN VÀ ĐỀ NGHỊ

5.1 Kết luận

5.2 Đề nghị

CHƯƠNG 2 : TỔNG QUAN TÀI LIỆU VÀ CƠ SỞ LÝ THUYẾT

2.1 Tổng quan về tối ưu tổ hợp

Tối ưu tổ hợp là một lĩnh vực quan trọng trong toán học và khoa học máy tính, chuyên nghiên cứu việc tìm kiếm giải pháp tốt nhất từ một tập hợp các khả năng có thể cho các bài toán có cấu trúc tổ hợp. Các bài toán này thường liên quan đến việc xác định giải pháp tối ưu hoặc gần tối ưu trong một không gian giải pháp rất lớn và phức tạp. Không gian giải pháp của các bài toán tổ hợp thường được tổ chức dưới dạng tập hợp, chuỗi, đồ thị, hoặc cấu trúc tổ hợp khác, và việc tìm kiếm giải pháp yêu cầu phải duyệt qua nhiều khả năng khác nhau.

Một số bài toán tối ưu tổ hợp tiêu biểu bao gồm:

- **Bài toán Tối ưu hóa Đồ thị:** Bao gồm các bài toán như tìm kiếm đường đi ngắn nhất trong đồ thị, tìm cây bao trùm nhỏ nhất, và tìm đường đi Hamiltonian. Ví dụ, bài toán tìm đường đi ngắn nhất (Shortest Path Problem) yêu cầu xác định đường đi ngắn nhất từ một điểm đến một điểm khác trong đồ thị, điều này có ứng dụng quan trọng trong hệ thống định vị và mạng lưới giao thông.
- **Bài toán Tối ưu hóa Định tuyến:** Ví dụ, bài toán tối ưu hóa định tuyến xe tải (Vehicle Routing Problem - VRP) nhằm tối ưu hóa lộ trình của các phương tiện vận chuyển để phục vụ khách hàng với chi phí thấp nhất. Đây là bài toán quan trọng trong logistics và quản lý chuỗi cung ứng.
- **Bài toán Tối ưu hóa Tổ hợp:** Các bài toán như bài toán đóng gói (Knapsack Problem) và bài toán sắp xếp (Scheduling Problems). Bài toán đóng gói yêu cầu xác định cách chọn một tập hợp con các đối tượng sao cho tổng giá trị của chúng là tối đa trong khi không vượt quá trọng lượng hoặc thể tích cho phép. Bài toán sắp xếp liên quan đến việc lập lịch các nhiệm vụ sao cho hoàn thành trong thời gian tối ưu và đáp ứng các yêu cầu hạn chế.

Tối ưu tổ hợp thường đối mặt với các thách thức do kích thước không gian giải pháp lớn và sự phức tạp của các bài toán. Nhiều bài toán trong tối ưu tổ hợp thuộc

lớp NP-kho, có nghĩa là không có thuật toán hiệu quả nào có thể giải quyết tất cả các trường hợp trong thời gian hợp lý. Điều này dẫn đến việc phát triển và áp dụng các phương pháp giải quyết khác nhau để tìm kiếm giải pháp tối ưu hoặc gần tối ưu.

Các phương pháp giải quyết chính bao gồm:

- **Thuật toán Tinh chỉnh:** Các phương pháp như tìm kiếm địa phương (Local Search) và tinh chỉnh toàn cầu (Global Search) được sử dụng để cải thiện giải pháp hiện tại bằng cách thay đổi từng yếu tố trong không gian giải pháp. Thuật toán tinh chỉnh địa phương, chẳng hạn như thuật toán hồi tiếc (Tabu Search), nhằm tìm kiếm giải pháp tốt hơn bằng cách điều chỉnh các yếu tố nhỏ trong giải pháp hiện tại.

- **Thuật toán Cây:** Bao gồm các phương pháp như cây quyết định (Decision Trees) và cây bao trùm nhỏ nhất (Minimum Spanning Trees). Thuật toán cây giúp giải quyết các bài toán có cấu trúc cây hoặc đồ thị, nơi mục tiêu là tìm kiếm cấu trúc tối ưu với chi phí thấp nhất.

- **Thuật toán Metaheuristic:** Các phương pháp như thuật toán di truyền (Genetic Algorithms), thuật toán đàn kiến (Ant Colony Optimization - ACO), và thuật toán simulated annealing. Những thuật toán này không đảm bảo tìm ra giải pháp tối ưu nhưng có thể tìm ra giải pháp gần tối ưu trong thời gian hợp lý. Chúng sử dụng các cơ chế như chọn lọc, lai ghép, đột biến trong thuật toán di truyền, hoặc mô phỏng hành vi của đàn kiến để tối ưu hóa quá trình tìm kiếm.

Tối ưu tổ hợp không chỉ có ý nghĩa lý thuyết mà còn có ứng dụng thực tiễn rộng rãi trong nhiều lĩnh vực. Trong kỹ thuật, tối ưu tổ hợp giúp cải thiện thiết kế, lập kế hoạch dự án, và tối ưu hóa quy trình sản xuất. Trong kinh doanh và logistics, nó hỗ trợ trong việc quản lý chuỗi cung ứng, tối ưu hóa lộ trình vận tải, và phân phối hàng hóa. Trong khoa học máy tính, tối ưu tổ hợp hỗ trợ phát triển thuật toán và cấu trúc dữ liệu hiệu quả. Trong kinh tế học, nó giúp mô hình hóa và phân tích các quyết định kinh tế trong điều kiện hạn chế, cải thiện khả năng ra quyết định và quản lý tài nguyên.

Nhờ vào sự phát triển của các phương pháp và công cụ tối ưu tổ hợp, chúng ta có thể giải quyết các bài toán phức tạp hơn và nâng cao hiệu quả trong nhiều ứng dụng thực tiễn. Các nghiên cứu và ứng dụng trong lĩnh vực này không ngừng mở rộng, cung cấp các giải pháp sáng tạo cho các thách thức ngày càng tăng trong nhiều lĩnh vực khác nhau

2.2 Tổng quan về bài toán tìm kiếm gần nhất

Bài toán tìm kiếm gần nhất (Nearest Neighbor Search - NNS) là một trong những bài toán cơ bản trong khoa học máy tính và trí tuệ nhân tạo. Mục tiêu của NNS là xác định điểm hoặc đối tượng trong tập dữ liệu có vị trí gần nhất với một điểm cho trước theo một khoảng cách định trước, thường là khoảng cách Euclid. Bài toán này có nhiều ứng dụng quan trọng trong các lĩnh vực như nhận dạng mẫu, hệ thống gợi ý, phân tích dữ liệu, và đặc biệt là trong hệ thống thông tin địa lý (GIS) và các bài toán liên quan đến không gian.

NNS có thể được giải quyết bằng nhiều phương pháp khác nhau. Các phương pháp truyền thống như cây k-d (k-d tree), cây phủ (cover tree), và phương pháp tìm kiếm tuyến tính đều có những ưu điểm và nhược điểm riêng. Tuy nhiên, khi kích thước dữ liệu ngày càng lớn và phức tạp, những phương pháp này gặp khó khăn trong việc cân bằng giữa độ chính xác và thời gian tính toán. Điều này đã thúc đẩy sự ra đời của các thuật toán heuristic và metaheuristic như thuật toán đàn kiến (ACO), nhằm mục đích tối ưu hóa hiệu quả tìm kiếm trong các không gian lớn

2.3 Giới thiệu về thuật toán đàn kiến (ACO)

Thuật toán đàn kiến (Ant Colony Optimization - ACO) được giới thiệu lần đầu tiên bởi Marco Dorigo vào những năm 1990 như một phương pháp metaheuristic để giải quyết các bài toán tối ưu hóa tổ hợp. ACO dựa trên việc mô phỏng hành vi tìm kiếm thức ăn của loài kiến trong tự nhiên. Khi tìm đường đến nguồn thức ăn, các con kiến để lại dấu vết pheromone trên đường đi của chúng. Những con kiến khác có xu hướng đi theo các dấu vết này, và các đường có lượng pheromone cao

hơn sẽ được ưu tiên. Qua thời gian, các con kiến dần tìm ra con đường ngắn nhất nhờ vào sự tự tổ chức và phản hồi tích cực này.

ACO đã được áp dụng thành công trong nhiều bài toán tối ưu hóa như bài toán người du lịch (Traveling Salesman Problem - TSP), bài toán phân chia mạng (network partitioning), và nhiều vấn đề khác liên quan đến định tuyến và lộ trình. Điểm mạnh của ACO nằm ở khả năng tự tổ chức, phân tán, và khả năng thích nghi linh hoạt với các thay đổi trong môi trường. Những đặc tính này khiến ACO trở thành một công cụ mạnh mẽ để giải quyết các bài toán tối ưu hóa trong không gian lớn và phức tạp, chẳng hạn như bài toán tìm kiếm gần nhất.

2.4 Liên hệ giữa ACO và bài toán tìm kiếm gần nhất

Ứng dụng của ACO vào bài toán tìm kiếm gần nhất dựa trên sự tương đồng giữa quá trình tìm kiếm đường ngắn nhất của loài kiến và quá trình tìm kiếm đối tượng gần nhất trong một tập hợp dữ liệu lớn. Khi áp dụng ACO vào NNS, mỗi con kiến có thể được coi như một tác tử tìm kiếm, di chuyển trong không gian dữ liệu và để lại dấu vết pheromone khi xác định các điểm gần với mục tiêu. Quá trình này được lặp đi lặp lại với mục đích tìm ra điểm gần nhất trong không gian một cách hiệu quả và tối ưu nhất.

Một trong những lợi ích quan trọng của việc áp dụng ACO vào NNS là khả năng tìm kiếm đồng thời trên nhiều hướng khác nhau, giảm thiểu rủi ro bị mắc kẹt ở các giải pháp cục bộ và tăng cơ hội tìm ra giải pháp tối ưu toàn cục. Hơn nữa, ACO có thể được kết hợp với các phương pháp khác như phân cụm (clustering) hoặc giảm chiều dữ liệu (dimensionality reduction) để cải thiện hiệu suất tìm kiếm trong các không gian dữ liệu lớn và phức tạp.

2.5 Ứng dụng ACO trong tìm kiếm gần nhất

Thuật toán đàn kiến (ACO) được áp dụng vào bài toán tìm kiếm gần nhất bằng cách sử dụng một mô hình mô phỏng hành vi của đàn kiến trong việc tìm kiếm đường đi. ACO dựa trên các khái niệm cơ bản như pheromone, thông tin heuristic và các quy tắc lựa chọn. Trong quá trình tìm kiếm, pheromone được cập nhật để hướng dẫn các con kiến tìm kiếm điểm gần nhất. Thông tin heuristic giúp cải thiện

hiệu quả của thuật toán bằng cách cung cấp thông tin bổ sung về khoảng cách giữa các điểm. Các bước chính của thuật toán bao gồm khởi tạo pheromone và các thông số, thực hiện tìm kiếm và cập nhật pheromone dựa trên kết quả tìm kiếm, và lặp lại quá trình cho đến khi đạt được tiêu chí dừng. Việc áp dụng ACO giúp cải thiện hiệu suất của bài toán tìm kiếm gần nhất bằng cách tối ưu hóa quá trình tìm kiếm và cập nhật thông tin.

2.6 Cách tiếp cận

Trên đây cho thấy các bài toán TỰTH có thể đưa về bài toán tìm kiếm trên đồ thị. Các bài toán này có thể giải đúng hoặc gần đúng. Với những bài toán cỡ nhỏ hoặc có dạng đặc biệt người ta có thể tìm lời giải tối ưu nhờ tìm kiếm vét cạn hoặc bằng một thuật toán với thời gian đa thức, được xây dựng dựa trên các phân tích toán học. Nhiều bài toán trong số đó là NP-khó, nên với các bài toán cỡ lớn, người ta phải tìm lời giải gần đúng. Các thuật toán giải gần đúng các bài toán TỰTH khó thường dựa trên 3 kỹ thuật cơ bản: heuristic cấu trúc (construction heuristic), tìm kiếm cục bộ (local search) và Metaheuristic.

2.6.1 Heuristic cấu trúc

Khi không thể tìm được lời giải tối ưu của bài toán, trong thực hành người ta tìm lời giải gần đúng. Một kỹ thuật hay được dùng là heuristic cấu trúc, trong đó lời giải của bài toán TỰTH được xây dựng theo cách mở rộng tuần tự. Từ thành phần khởi tạo trong tập C_0 ở mục 1.1, từng bước mở rộng không quay lui, bằng cách thêm vào các thành phần mới theo phương thức ngẫu nhiên hay tất định dựa trên các quy tắc heuristic đã chọn. Các quy tắc heuristic này thường được xây dựng dựa trên các kết quả phân tích toán học hoặc kinh nghiệm. Phương pháp heuristic cấu trúc tham khảo sau đây cho ta hình dung được cách tiếp cận này (Hình 2.1).

```

Procedure Heuristic cấu trúc tham ăn;
Begin
     $s_p \rightarrow$  chọn thành phần  $u_0$  trong  $C_0$ ;
    while (chưa xây dựng xong lời giải) do
         $c \rightarrow \text{GreedyComponent}(s_p)$ ;
         $s_p \rightarrow s_p \wedge c$ ;
    end-while
     $s \rightarrow s_p$ ;
    Đưa ra lời giải  $s$ ;
End;

```

Hình 2.1 Phương pháp heuristic cấu trúc tham ăn

Trong đó $\text{GreedyComponent}(s_p)$ có nghĩa là chọn thành phần bổ sung vào s_p theo quy tắc heuristic đã có. Ký hiệu $s_p \wedge c$ là kết quả phép toán thêm thành phần c vào s_p .

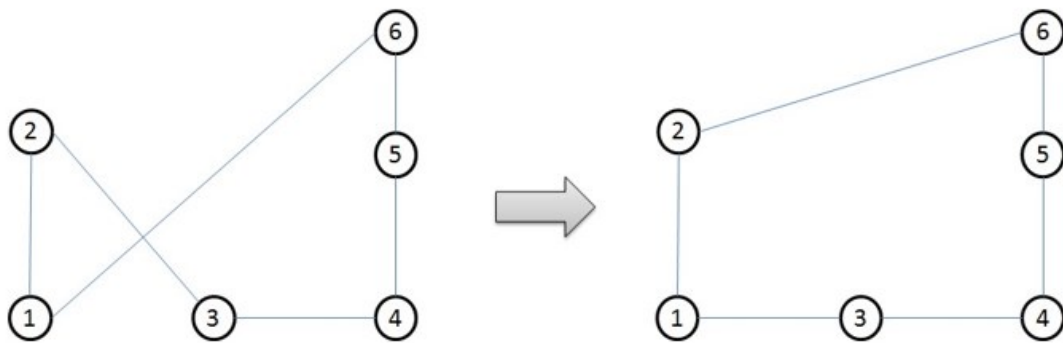
Dễ dàng hình dung phương pháp này khi áp dụng thuật toán cho bài toán TSP với đồ thị đầy đủ và sử dụng quy tắc heuristic *láng giềng gần nhất để chọn đỉnh thêm vào* (tức là chọn đỉnh gần nhất chưa đi qua để thêm vào hành trình). Các thuật toán này có ưu điểm là tốn ít thời gian chạy nhưng nhược điểm chính là không cải tiến lời giải được.

2.6.2 Tìm kiếm cục bộ (local search).

Kỹ thuật tìm kiếm cục bộ hay còn gọi là tìm kiếm địa phương, thực hiện bằng cách bắt đầu từ một phương án chấp nhận được, lặp lại bước cải tiến lời giải nhờ các thay đổi cục bộ. Để thực hiện kỹ thuật này, ta cần xác định được cấu trúc lân cận của mỗi phương án (lời giải) đang xét, tức là những phương án chấp nhận được, gần với nó nhất, nhờ thay đổi một số thành phần. Cách thường dùng là lân cận k -thay đổi, tức là lân cận bao gồm các phương án chấp nhận được khác với phương án đang xét nhờ thay đổi nhiều nhất k thành phần.

Ví dụ. Lân cận 2-thay đổi của một lời giải s trong bài toán TSP bao gồm tất cả các lời giải s có thể nhận được từ s bằng cách đổi hai cạnh. Hình 2.2 chỉ ra một ví dụ một lời giải nhận được bằng cách thay hai cạnh (2,3), (1,6) bằng hai cạnh (1,3), (2,6).

Việc cải tiến trong các bước lặp thường chọn theo phương pháp leo đồi dựa theo hai chiến lược: Chiến lược tốt nhất và chiến lược tốt hơn. Với chiến lược tốt nhất, người ta thực hiện chọn lời giải tốt nhất trong lân cận để làm lời giải cải tiến. Tuy nhiên, khi bài toán cỡ lớn có thể không tìm được lời giải tốt nhất do bị hạn chế về thời gian. Còn với chiến lược tốt hơn, ta chọn phương án đầu tiên trong lân cận, cải thiện được hàm mục tiêu. Nhược điểm của tìm kiếm cục bộ là thường chỉ cho cực trị địa phương.



Hình 2.2 Lời giải nhận được nhờ thay 2 cạnh $(2,3)$, $(1,6)$ bằng $(1,3)$, $(2,6)$

Các kỹ thuật trên thường được kết hợp, tạo thành các hệ lai trong các phương pháp mô phỏng tự nhiên dựa trên quần thể, chẳng hạn như thuật toán di truyền (GA) hoặc tối ưu đàn kiến (ACO).

2.6.3 Metaheuristic.

Phương pháp metaheuristic là một phương pháp heuristic tổng quát được thiết kế, định hướng cho các thuật toán cụ thể (bao gồm cả heuristic cấu trúc và tìm kiếm cục bộ). Như vậy, một metaheuristic là một lược đồ thuật toán tổng quát ứng dụng cho các bài toán tối ưu khác nhau, với một chút sửa đổi cho phù hợp với từng bài toán.

Memetic là một mô hình theo phương pháp metaheuristic. Trong các thuật toán được thiết kế theo memetic, người ta tạo ra nhiều thể hệ quần thể lời giải chấp

nhận được. Trong mỗi quần thể của thể hệ tương ứng, ta chỉ chọn ra một số lời giải (chẳng hạn lời giải tốt nhất) để thực hiện tìm kiếm cục bộ nhằm cải thiện chất lượng. Quá trình tiến hóa này cho ta tìm được lời giải tốt nhất có thể. Hình 2.3 mô tả một thuật toán memetic sử dụng tính toán tiến hóa (*Evolutionary Computing - EC*):

Trong ứng dụng thực tế, các thuật toán ACO thường được kết hợp với tìm kiếm cục bộ theo mô hình memetic này.

```

Proedure Thuật toán memetic-EC;
Begin
  Initialize: Tạo ra quần thể đầu tiên;
  while điều kiện dừng chưa thỏa mãn do
    Đánh giá các cá thể trong quần thể;
    Thực hiện tiến hóa quần thể nhờ các toán tử cho trước;
    Chọn tập con  $f_{il}$  để cải tiến nhờ thủ tục tìm kiếm cục bộ;
    for mỗi cá thể trong  $f_{il}$  do
      Thực hiện tìm kiếm cục bộ;
    end-for
    Chọn phần tử tốt nhất;
  end-while;
  Đưa ra lời giải tốt nhất;
End;

```

Hình 2.3 Thuật toán memetic sử dụng EC

2.6.4 Công cụ và phương pháp

Để thực hiện nghiên cứu, chúng tôi sử dụng Python cùng với các thư viện như NumPy và SciPy cho tính toán số học, và Matplotlib hoặc Seaborn cho việc trực quan hóa kết quả. Môi trường phát triển chính là Jupyter Notebook, nhờ vào khả năng tương tác và hỗ trợ việc thử nghiệm nhanh chóng. Các công cụ thử nghiệm bao gồm các thư viện thống kê và phần mềm mô phỏng, giúp đánh giá hiệu suất của thuật toán ACO. Việc lựa chọn các công cụ này giúp đảm bảo tính chính xác và hiệu quả trong quá trình phát triển và thử nghiệm thuật toán.

2.6.5 Thiết kế thí nghiệm và quy trình thực hiện nghiên cứu

Thiết kế thí nghiệm sẽ bao gồm việc lựa chọn các tập dữ liệu thử nghiệm, chẳng hạn như các tập điểm trong không gian hai chiều hoặc ba chiều. Các kịch bản thí nghiệm sẽ được thiết lập để kiểm tra hiệu suất của thuật toán ACO trong nhiều

điều kiện khác nhau. Tiêu chí đánh giá kết quả thí nghiệm bao gồm thời gian tính toán, độ chính xác của kết quả và khả năng mở rộng của thuật toán. Phương pháp đo lường sẽ so sánh hiệu suất của ACO với các phương pháp tìm kiếm gần nhất khác và phân tích các kết quả thu được. Kết quả dự kiến sẽ cung cấp cái nhìn sâu sắc về hiệu quả của thuật toán ACO trong việc giải quyết bài toán tìm kiếm gần nhất và đưa ra những kết luận có thể từ thí nghiệm.

2.6.6 Thuật toán Rank-Based Ant System (RBAS)

Đây cũng là một thuật toán được mở rộng phát triển từ hệ thống AS đưa ra bởi Bullnheimer, Hartl và Strauss vào năm 1997. Thuật toán này đưa vào ý tưởng xếp hạng cho các lời giải khi thực hiện cập nhật pheromone. Cụ thể như sau:

- Đầu tiên, m con kiến được xếp hạng theo thứ tự giảm dần dựa theo chất lượng lời giải mà nó thu được. Ví dụ: $(S_1, S_2, \dots, S_{m-1}, S_m)$ trong đó S_1 là phương án tốt nhất.
- Pheromone chỉ được đặt thêm trên các cung của $G - 1$ con kiến có lời giải tốt nhất. Lượng pheromone cũng phụ thuộc trực tiếp vào thứ hạng sắp xếp của con kiến.
- Các đoạn đường đi của lời giải tốt nhất được nhận thêm một lượng pheromone phụ thuộc vào chất lượng lời giải.

2.6.7 Các nghiên cứu liên quan

Trong những năm qua, nhiều nghiên cứu đã được thực hiện nhằm phát triển và ứng dụng ACO trong các bài toán tối ưu hóa khác nhau. Một số nghiên cứu đã chỉ ra rằng ACO có thể vượt trội hơn so với các phương pháp truyền thống trong các bài toán tìm kiếm và tối ưu hóa. Ví dụ, việc áp dụng ACO vào bài toán người du lịch đã chứng minh khả năng của thuật toán trong việc tìm ra các lộ trình ngắn nhất trong các không gian tìm kiếm lớn. Tương tự, một số nghiên cứu khác đã áp dụng ACO vào bài toán tìm kiếm gần nhất và thu được kết quả khả quan, đặc biệt là trong các hệ thống có dữ liệu không đồng nhất và phức tạp.

Tuy nhiên, cũng cần lưu ý rằng việc triển khai ACO không phải lúc nào cũng dễ dàng và hiệu quả phụ thuộc vào nhiều yếu tố như tham số của thuật toán, cấu trúc

của không gian dữ liệu, và yêu cầu cụ thể của bài toán. Do đó, việc nghiên cứu kỹ lưỡng và điều chỉnh thuật toán sao cho phù hợp với bài toán cụ thể là rất quan trọng.

CHƯƠNG 3 : PHƯƠNG PHÁP THỰC HIỆN

3.1 Mô tả bài toán

Bài toán tìm kiếm gần nhất (Nearest Neighbor Search) yêu cầu xác định điểm gần nhất trong một tập hợp điểm cho trước đối với một điểm nguồn. Định nghĩa bài toán này bao gồm việc tìm điểm trong tập hợp sao cho khoảng cách từ điểm nguồn đến điểm đó là nhỏ nhất. Bài toán này có thể được áp dụng trong nhiều lĩnh vực thực tiễn như hệ thống định vị, lọc thông tin, và các hệ thống khuyến nghị. Kích thước của tập hợp điểm, số lượng điểm, và các yêu cầu về thời gian hoặc không gian là các yếu tố quan trọng trong việc giải quyết bài toán này. Việc xác định và mô hình hóa bài toán một cách chính xác là bước đầu tiên để áp dụng các thuật toán tối ưu, chẳng hạn như thuật toán đàn kiến (ACO), nhằm cải thiện hiệu suất tìm kiếm.

3.2 Ứng dụng ACO trong tìm kiếm gần nhất

Thuật toán đàn kiến (ACO) được áp dụng vào bài toán tìm kiếm gần nhất bằng cách sử dụng một mô hình mô phỏng hành vi của đàn kiến trong việc tìm kiếm đường đi. ACO dựa trên các khái niệm cơ bản như pheromone, thông tin heuristic và các quy tắc lựa chọn. Trong quá trình tìm kiếm, pheromone được cập nhật để hướng dẫn các con kiến tìm kiếm điểm gần nhất. Thông tin heuristic giúp cải thiện hiệu quả của thuật toán bằng cách cung cấp thông tin bổ sung về khoảng cách giữa các điểm. Các bước chính của thuật toán bao gồm khởi tạo pheromone và các thông số, thực hiện tìm kiếm và cập nhật pheromone dựa trên kết quả tìm kiếm, và lặp lại quá trình cho đến khi đạt được tiêu chí dừng. Việc áp dụng ACO giúp cải thiện hiệu suất của bài toán tìm kiếm gần nhất bằng cách tối ưu hóa quá trình tìm kiếm và cập nhật thông tin. Việc đó có thể được thực hiện qua các bước sau:

1. Khởi tạo

- Tập hợp các điểm trong không gian cần tìm.

- Xác định số lượng "kiến" ban đầu (mỗi kiến là một agent bắt đầu từ một điểm ngẫu nhiên).
- Khởi tạo các thông số của thuật toán: số lần lặp lại, mức độ pheromone, tốc độ bay hơi pheromone, và khả năng heuristic (hàm khoảng cách giữa các điểm).

2. Xây dựng lời giải

- Mỗi con kiến bắt đầu từ một điểm và sẽ chọn điểm tiếp theo dựa trên lượng pheromone và khoảng cách đến các điểm lân cận.
- Lựa chọn điểm tiếp theo bằng công thức xác suất, trong đó bao gồm cả tác động của pheromone và yếu tố heuristic (khoảng cách):

$$P_{ij} = \frac{(\tau_{ij})^{\alpha} (\eta_{ij})^{\beta}}{\sum_{k \in \text{not visited}} (\tau_{ik})^{\alpha} (\eta_{ik})^{\beta}}$$

Hình 3.1 Công thức tính xác suất lựa chọn

Trong đó:

- + P_{ij} : Xác suất lựa chọn từ điểm i đến điểm j .
- + τ_{ij} : Lượng pheromone trên đường từ điểm i đến j .
- + η_{ij} : Thông tin heuristic (khoảng cách nghịch đảo giữa i và j).
- + α, β : Các thông số điều khiển tầm quan trọng của pheromone và thông tin heuristic.

3. Cập nhật pheromone

- + Sau khi tất cả kiến đã hoàn thành việc tìm kiếm một điểm gần nhất, các kiến sẽ để lại dấu vết pheromone trên các đường đi mà chúng đã chọn.
- + Pheromone sẽ được cập nhật theo công thức:

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

Hình 3.2 Công thức tính lượng pheromone

- + ρ : Tỷ lệ bay hơi pheromone
- + $\Delta\tau_{ij}$: Lượng pheromone do kiến thứ k để lại.

4. Cập nhật pheromone

- + Quá trình tìm kiếm và cập nhật pheromone sẽ được lặp lại cho đến khi đạt được kết quả tốt hoặc đạt ngưỡng số lần lặp lại nhất định.
- + Kiến sẽ dần ưu tiên các đường có lượng pheromone lớn hơn và khả năng heuristic tốt hơn, giúp tìm kiếm các điểm gần nhất.

5. Kết quả

- + Sau nhiều lần lặp lại, các con kiến sẽ dần tối ưu hóa hành trình tìm kiếm các điểm gần nhất, từ đó tìm ra những điểm gần nhau trong không gian với độ chính xác cao.

3.3 Ứng dụng trong tìm kiếm điểm gần nhất

- ACO có thể được áp dụng để tìm các điểm gần nhau trong không gian như bài toán Travelling Salesman Problem (TSP) hoặc bất kỳ bài toán tối ưu hóa liên quan đến tìm kiếm khoảng cách ngắn nhất giữa các điểm.
- Cụ thể, nếu bạn có một tập các điểm trong không gian và cần tìm tập hợp các điểm gần nhau nhất, ACO sẽ tối ưu hóa quá trình lựa chọn dựa

trên hành vi tìm đường và đánh giá các yếu tố như khoảng cách giữa các điểm.

Cụ thể mô tả bài toán tìm kiếm gần nhất sẽ gồm những điểm sau:

- + Cho một tập các tọa độ điểm trên bản đồ (các điểm có thể là kinh độ và vĩ độ).
- + Xác định một tọa độ điểm trung tâm.
- + Tìm những điểm gần nhất với tọa độ trung tâm này dựa trên khoảng cách.

3.4 Giải pháp:

- + Sử dụng khoảng cách Euclidean để tính khoảng cách giữa các điểm.
 - + Xếp hạng các điểm dựa trên khoảng cách gần nhất.
- + Trả về các điểm gần nhất.

Ở đây có 2 cách áp dụng giải bài tìm gần nhất trên:

- + Áp dụng thuật toán NNS (Nearest Neighbor Search)

```

1 import math
2
3 # Hàm tính khoảng cách Euclidean giữa 2 điểm (x1, y1) và (x2, y2)
4 def euclidean_distance(point1, point2):
5     return math.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)
6
7 # Tìm n điểm gần nhất với điểm xác định
8 def find_nearest_points(reference_point, points, n=5):
9     # Tính khoảng cách từ điểm xác định tới tất cả các điểm
10    distances = [(point, euclidean_distance(reference_point, point)) for point in points]
11
12    # Sắp xếp các điểm theo khoảng cách
13    distances.sort(key=lambda x: x[1])
14
15    # Trả về n điểm gần nhất
16    return [point for point, dist in distances[:n]]
17
18 # Danh sách các tọa độ điểm trên bản đồ
19 points = [
20     (10.7769, 106.7009), # Tọa độ TP HCM
21     (21.0285, 105.8542), # Tọa độ Hà Nội
22     (16.0471, 108.2062), # Tọa độ Đà Nẵng
23     (10.8231, 106.6297), # Một tọa độ khác tại TP HCM
24     (11.9416, 108.4583), # Tọa độ Đà Lạt
25 ]
26
27 # Tọa độ điểm tham chiếu (ví dụ TP HCM)
28 reference_point = (10.7769, 106.7009)
29
30 # Số lượng điểm cần tìm
31 n = 3
32
33 # Tìm n điểm gần nhất
34 nearest_points = find_nearest_points(reference_point, points, n)
35
36 # In kết quả
37 print(f"Các điểm gần nhất với {reference_point} là:")
38 for i, point in enumerate(nearest_points, 1):
39     print(f"{i}. {point}")

```

Hình 3.3 Thuật toán NNS

Với danh sách các điểm trên bản đồ và điểm tham chiếu là tọa độ TP HCM, chương trình sẽ trả về các điểm gần nhất như sau:

```

Các điểm gần nhất với (10.7769, 106.7009) là:
1. (10.7769, 106.7009)
2. (10.8231, 106.6297)
3. (11.9416, 108.4583)

```

Hình 3.4 Kết quả các điểm gần nhất

+ Áp dụng thuật toán ACO

```

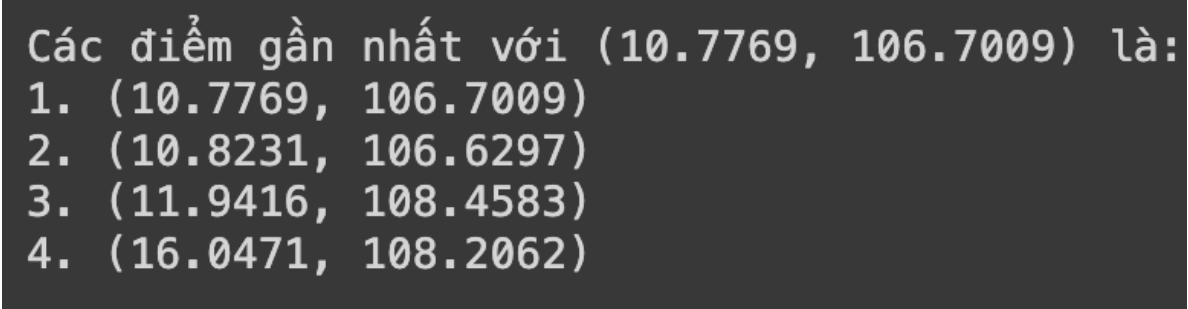
1 import random
2 import math
3
4 # Hàm tính khoảng cách Euclidean giữa 2 điểm (x1, y1) và (x2, y2)
5 def euclidean_distance(point1, point2):
6     return math.sqrt((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)
7
8 # Hàm ACO để tìm các điểm gần nhất
9 class ACO:
10     def __init__(self, points, alpha=1.0, beta=2.0, evaporation_rate=0.5, num_ants=10, num_iterations=100):
11         self.points = points # Danh sách các điểm trên bản đồ
12         self.num_points = len(points)
13         self.alpha = alpha # Hệ số pheromone
14         self.beta = beta # Hệ số heuristic
15         self.evaporation_rate = evaporation_rate # Tỷ lệ bay hơi pheromone
16         self.num_ants = num_ants # Số lượng kiến
17         self.num_iterations = num_iterations # Số lần lặp lại thuật toán
18         self.pheromone = [[1 for _ in range(self.num_points)] for _ in range(self.num_points)] # Khởi tạo pheromone
19
20     def probability(self, i, j, heuristic):
21         # Tính xác suất kiến chọn từ điểm i đến điểm j
22         return (self.pheromone[i][j]**self.alpha) * (heuristic[i][j]**self.beta)
23
24     def update_pheromone(self, all_ants_paths, all_ants_lengths):
25         # Giảm pheromone cũ (bay hơi)
26         for i in range(self.num_points):
27             for j in range(self.num_points):
28                 self.pheromone[i][j] *= (1 - self.evaporation_rate)
29
30         # Cập nhật pheromone dựa trên hành trình của kiến
31         for ant in range(self.num_ants):
32             for i in range(len(all_ants_paths[ant]) - 1):
33                 from_point = all_ants_paths[ant][i]
34                 to_point = all_ants_paths[ant][i + 1]
35                 self.pheromone[from_point][to_point] += 1.0 / all_ants_lengths[ant]
36
37     def run(self, reference_point, n):
38         # Tính ma trận heuristic (khoảng cách nghịch đảo)
39         heuristic = [[1 / euclidean_distance(p1, p2) if euclidean_distance(p1, p2) != 0 else 0 for p2 in self.points] for p1 in self.points]
40
41         reference_index = self.points.index(reference_point)
42
43         for _ in range(self.num_iterations):
44             all_ants_paths = []
45             all_ants_lengths = []
46
47             # Mỗi kiến sẽ bắt đầu từ điểm tham chiếu
48             for _ in range(self.num_ants):
49                 current_index = reference_index
50                 visited = [current_index]
51
52                 for _ in range(n):
53                     probabilities = []
54                     total_prob = 0
55
56                     # Tính xác suất di chuyển tới các điểm chưa ghé qua
57                     for j in range(self.num_points):
58                         if j not in visited:
59                             prob = self.probability(current_index, j, heuristic)
60                             probabilities.append((j, prob))
61                             total_prob += prob
62
63                     # Chọn điểm tiếp theo dựa trên xác suất
64                     if total_prob > 0:
65                         next_index = random.choices(
66                             [point for point, _ in probabilities],
67                             [prob / total_prob for _, prob in probabilities]
68                             )[0]
69                     else:
70                         next_index = random.choice([i for i in range(self.num_points) if i not in visited])
71
72                     visited.append(next_index)
73                     current_index = next_index
74
75                     # Lưu hành trình và độ dài của mỗi kiến
76                     total_length = sum([euclidean_distance(self.points[visited[i]], self.points[visited[i + 1]]) for i in range(len(visited) - 1)])
77                     all_ants_paths.append(visited)
78                     all_ants_lengths.append(total_length)
79
80                     # Cập nhật pheromone sau khi tất cả kiến đã di chuyển
81                     self.update_pheromone(all_ants_paths, all_ants_lengths)
82
83                 # Tìm hành trình tốt nhất
84                 best_path = min(all_ants_paths, key=lambda path: sum([euclidean_distance(self.points[path[i]], self.points[path[i + 1]]) for i in range(len(path) - 1)]))
85                 return [self.points[i] for i in best_path]
86
87 # Tập các tọa độ điểm trên bản đồ (vĩ độ, kinh độ)
88 points = [
89     (10.7769, 106.7009), # Tọa độ TP HCM
90     (21.0285, 105.8542), # Tọa độ Hà Nội
91     (16.0471, 108.2062), # Tọa độ Đà Nẵng
92     (10.8231, 106.6297), # Một tọa độ khác tại TP HCM
93     (11.9416, 108.4583), # Tọa độ Đà Lạt
94 ]
95
96 # Tọa độ điểm tham chiếu
97 reference_point = (10.7769, 106.7009) # TP HCM
98
99 # Số lượng điểm gần nhất cần tìm
100 n = 3
101
102 # Khởi tạo ACO và chạy thuật toán
103 aco = ACO(points, num_ants=20, num_iterations=50)
104 nearest_points = aco.run(reference_point, n)
105
106 # In ra kết quả
107 print("Các điểm gần nhất với (reference_point) là:")
108 for i, point in enumerate(nearest_points, 1):
109     print(f"{i}. (point)")
110

```

Hình 3.5 Thuật toán ACO

Lớp ACO này thực hiện việc tối ưu tìm các điểm gần nhất. Các kiến sẽ di chuyển dựa trên xác suất phụ thuộc vào pheromone và khoảng cách các điểm (heuristic). Sau mỗi vòng lặp, pheromone sẽ được cập nhật để phản ánh các hành trình tốt

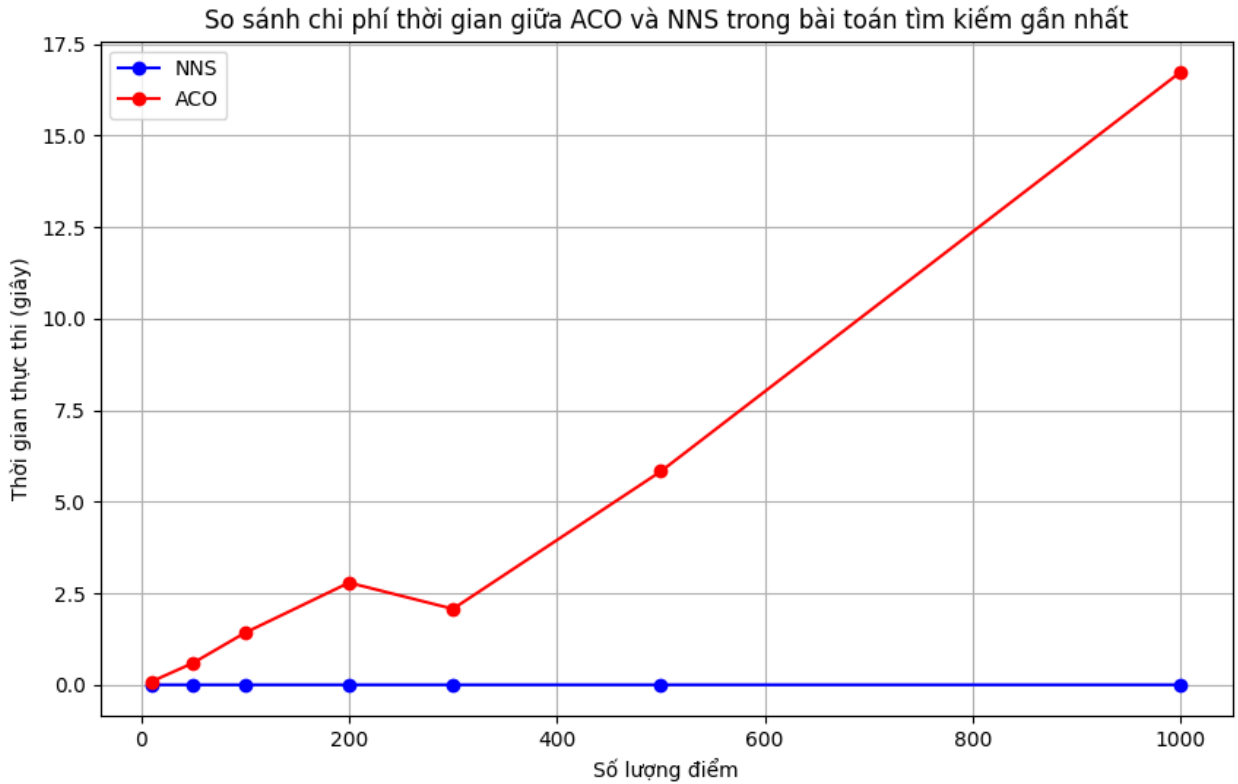
hơn. Với ACO, kiến bắt đầu từ điểm tham chiếu và cố gắng tìm các điểm gần nhất bằng cách tính xác suất lựa chọn điểm tiếp theo dựa trên lượng pheromone và khả năng heuristic. Kết quả qua nhiều lần lặp (iterations), thuật toán sẽ trả về danh sách các điểm gần nhất như bên dưới:



```
Các điểm gần nhất với (10.7769, 106.7009) là:  
1. (10.7769, 106.7009)  
2. (10.8231, 106.6297)  
3. (11.9416, 108.4583)  
4. (16.0471, 108.2062)
```

Hình 3.6 Kết quả các điểm gần nhất

Tuy cùng cách giải nhưng khi tăng số điểm lên thì hiệu quả và chi phí giữa 2 thuật toán ACO vs NNS rõ ràng khác biệt:



Hình 3.7 Biểu đồ so sánh chi phí thời gian giữa ACO và NNS

Trên là biểu đồ so sánh chi phí thời gian giữa 2 thuật toán ACO và NNS trong bài toán tìm kiếm điểm gần nhất. Biểu đồ cho thấy rằng:

- + Thuật toán NNS có xu hướng thực thi nhanh hơn, đặc biệt với số lượng điểm nhỏ đến vừa.
- + Thuật toán ACO tốn nhiều thời gian hơn, do cần nhiều lần lặp và cập nhật pheromone cho các kiến.

Rõ ràng ở bài toán số điểm lớn chúng ta nên chọn ACO. Nhưng bản thân ACO cũng tồn tại nhiều nhược điểm trên. Đó là sự ra đời của nhiều biến thể sau:

#1 Ant Colony System (ACS):

- + ACS là một biến thể quan trọng của ACO do Dorigo giới thiệu, nhằm tối ưu hoá quá trình cập nhật pheromone và lựa chọn hành trình của kiến.
- + Tối ưu hơn như thế nào:

++ Local pheromone update: Pheromone được cập nhật ngay sau khi mỗi con kiến di chuyển từ một điểm đến điểm khác, giúp làm giảm sự thu

hút quá mức của các con đường đã đi qua, khuyến khích kiến khám phá các con đường mới nhiều hơn.

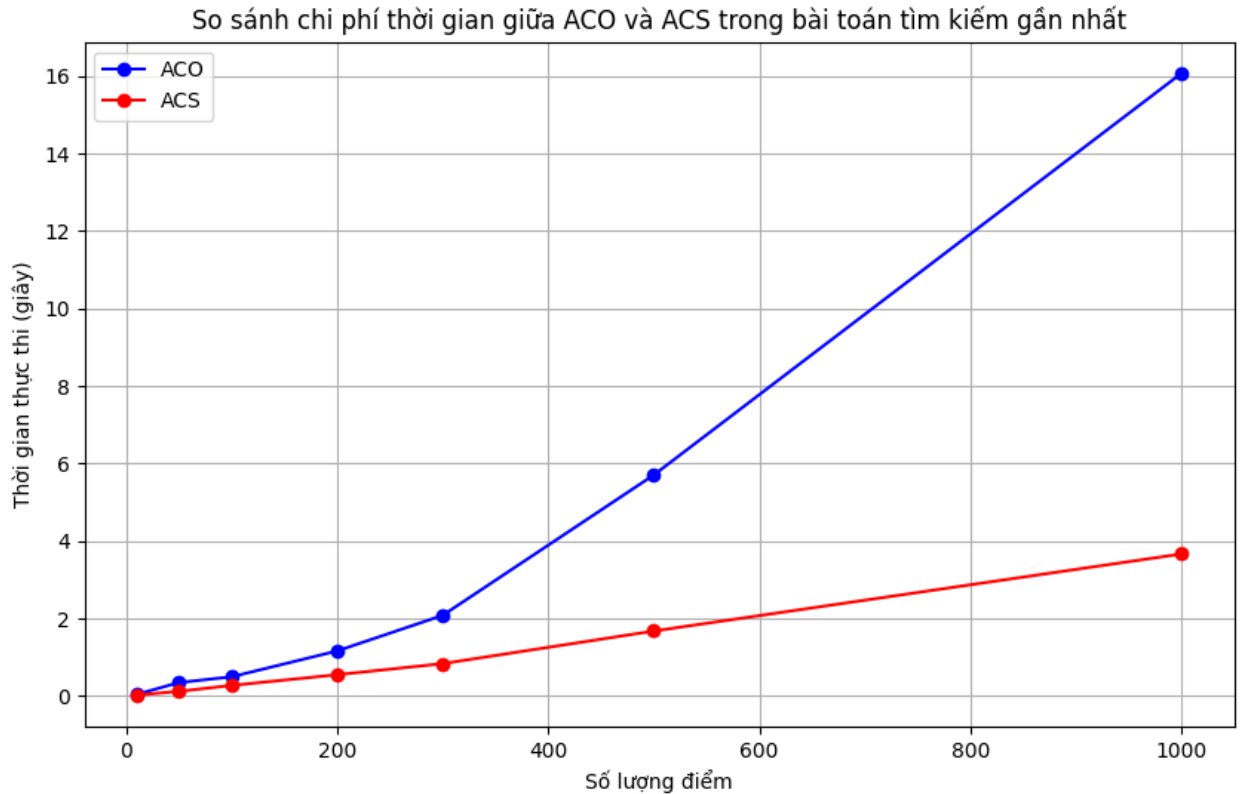
++ Global pheromone update: Chỉ kiến tìm được giải pháp tốt nhất (global-best solution) mới cập nhật pheromone, giúp tập trung tăng cường các giải pháp chất lượng cao.

++ Rule-based decision: Sử dụng công thức lựa chọn xác suất theo quy tắc

$$P_{ij} = (1 - \epsilon) \cdot \text{Exploration} + \epsilon \cdot \text{Exploitation}$$

Hình 3.8 Công thức tính xác suất lựa chọn P_{ij}

Trong đó, ϵ là hệ số cho phép kiến vừa khám phá hành trình mới (exploration) vừa khai thác hành trình cũ (exploitation). ACS giúp giảm thời gian tìm kiếm vì kiến có xu hướng tìm các hành trình tốt hơn nhanh hơn và không bị mắc kẹt trong các cụm bẫy cục bộ. Cụ thể ở cùng ví dụ trên ACS mang lại chi phí tốt hơn ACO



Hình 3.9 Biểu đồ so sánh chi phí thời gian giữa ACO và ACS

Từ biểu đồ, có thể thấy rằng ACS có hiệu suất tốt hơn ACO với thời gian thực thi thấp hơn, đặc biệt số lượng điểm tăng lên. Điều này là do cơ chế khai thác và cập nhật pheromone tối ưu hơn các ACS, giúp nó hội tụ nhanh hơn ACO thông thường.

#2 Max-Min Ant System (MMAS)

MMAS là một biến thể của ACO với sự thay đổi trong cách cập nhật và giới hạn giá trị pheromone.

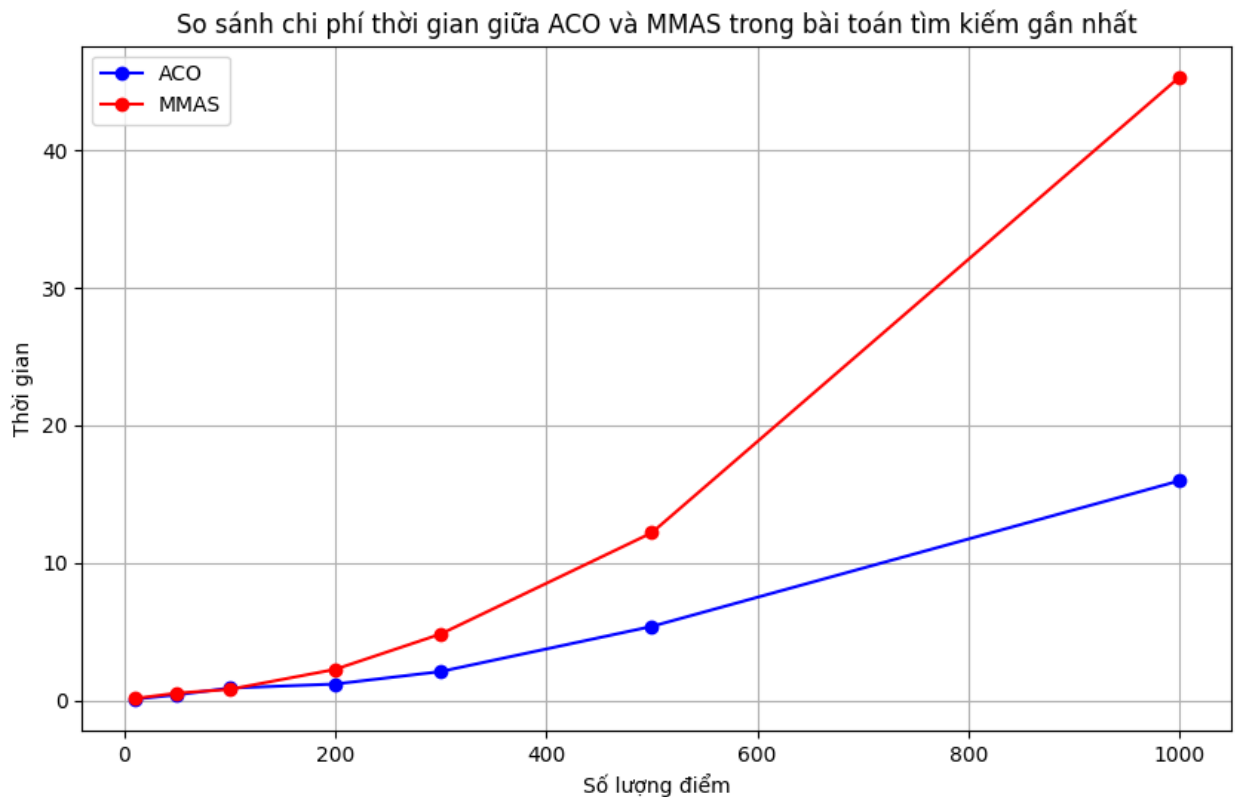
Tối ưu hơn như thế nào:

- + Giới hạn pheromone: Pheromone được giới hạn trong khoảng $[\tau_{min}, \tau_{max}]$, ngăn chặn việc pheromone tăng quá cao hoặc quá thấp, giúp kiến không quá phụ thuộc vào một hành trình cụ thể.

+ Cập nhật pheromone toàn cục: Chỉ cập nhật pheromone trên hành trình tốt nhất toàn cục hoặc tốt nhất trong lần lặp hiện tại. Điều này giúp các giải pháp tốt hơn được ưu tiên mạnh mẽ hơn.

+ Pheromone Evaporation: Tăng cường khả năng khám phá bằng cách cho phép pheromone trên các đường đi ít phổ biến bị bay hơi nhanh hơn, từ đó mở rộng khả năng khám phá của kiến.

MMAS giúp kiến khám phá nhiều hành trình khác nhau và giảm khả năng bị mắc kẹt trong pháp cục bộ, từ đó tăng hiệu quả tìm kiếm và cải thiện chất lượng giải pháp. Ta áp dụng lại cùng bài toán ở trên ra được kết quả như bên dưới:



Hình 3.10 Biểu đồ so sánh chi phí thời gian ACO và MMAS

Từ biểu đồ ta thấy sự khác biệt về hiệu suất giữa hai thuật toán khi số lượng điểm tăng lên

+ MMAS thường có xu hướng nhanh hơn ACO khi số lượng điểm tăng, nhờ cơ chế giới hạn pheromone và cập nhật thông minh hơn.

+ ACO mất nhiều thời gian hơn trong quá trình hội tụ khi làm việc với số lượng điểm lớn.

#3 Elitist Ant System (EAS)

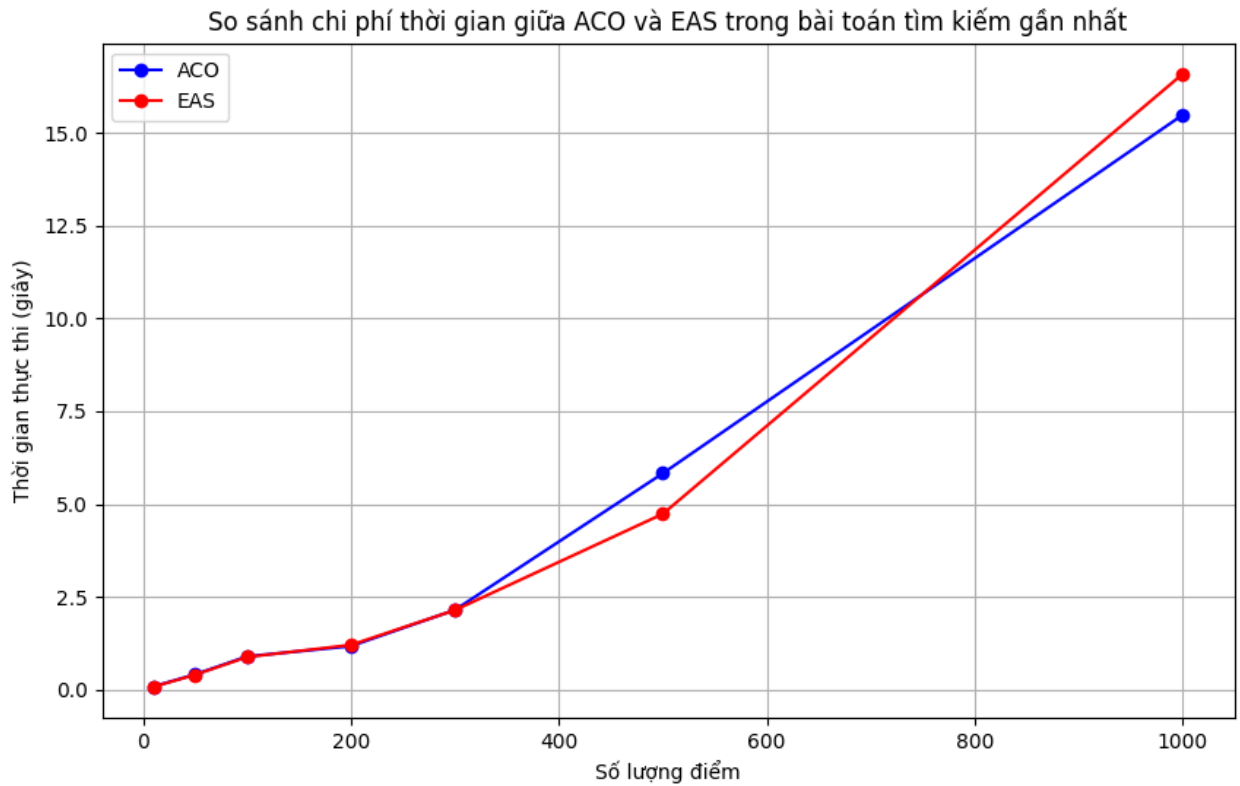
EAS là 1 biến thể của ACO trong đó kiến có giải pháp tốt nhất toàn cục (elite ant) được ưu tiên trong quá trình cập nhật pheromone.

Tối ưu hơn như thế nào:

+ Tăng cường pheromone cho kiến tốt nhất: Mỗi vòng lặp, kiến có giải pháp tốt nhất sẽ được cập nhật pheromone với trọng số lớn hơn so với các kiến thức, từ đó giúp các kiến khác dễ dàng học theo.

+ Hội tụ nhanh hơn: Do kiến tốt nhất được ưu tiên, các kiến khác sẽ nhanh chóng học theo và hội tụ về giải pháp tốt nhất toàn cục.

EAS giúp giảm thời gian tìm kiếm khi các kiến có xu hướng đi theo con đường của kiến tốt nhất. Tuy nhiên, hệ thống có thể nhanh chóng hội tụ vào các giải pháp cục bộ nếu không có chiến lược khai thác và khám phá hợp lý. Với cách xử lý trên ta áp dụng cùng 1 bài toán mang lại kết quả như bên dưới:



Hình 3.11 Biểu đồ so sánh chi phí thời gian ACO và EAS

Biểu đồ cũng cùng thể hiện hiệu suất khác biệt của giữa 2 thuật toán EAS vs ACO:

- + ACO có xu hướng mất nhiều thời gian hơn khi tập điểm lớn.
- + EAS với cơ chế cập nhật pheromone dựa trên kiến ưu tú giúp hội tụ nhanh hơn và tiết kiệm thời gian khi làm việc với số lượng điểm lớn.

#4 Rank-based Ant System (RAS)

RAS là 1 biến thể trong đó các con kiến được xếp hạng dựa trên chất lượng giải pháp chúng tìm được. Những kiến có thứ hạng cao sẽ được ưu tiên khi cập nhật pheromone.

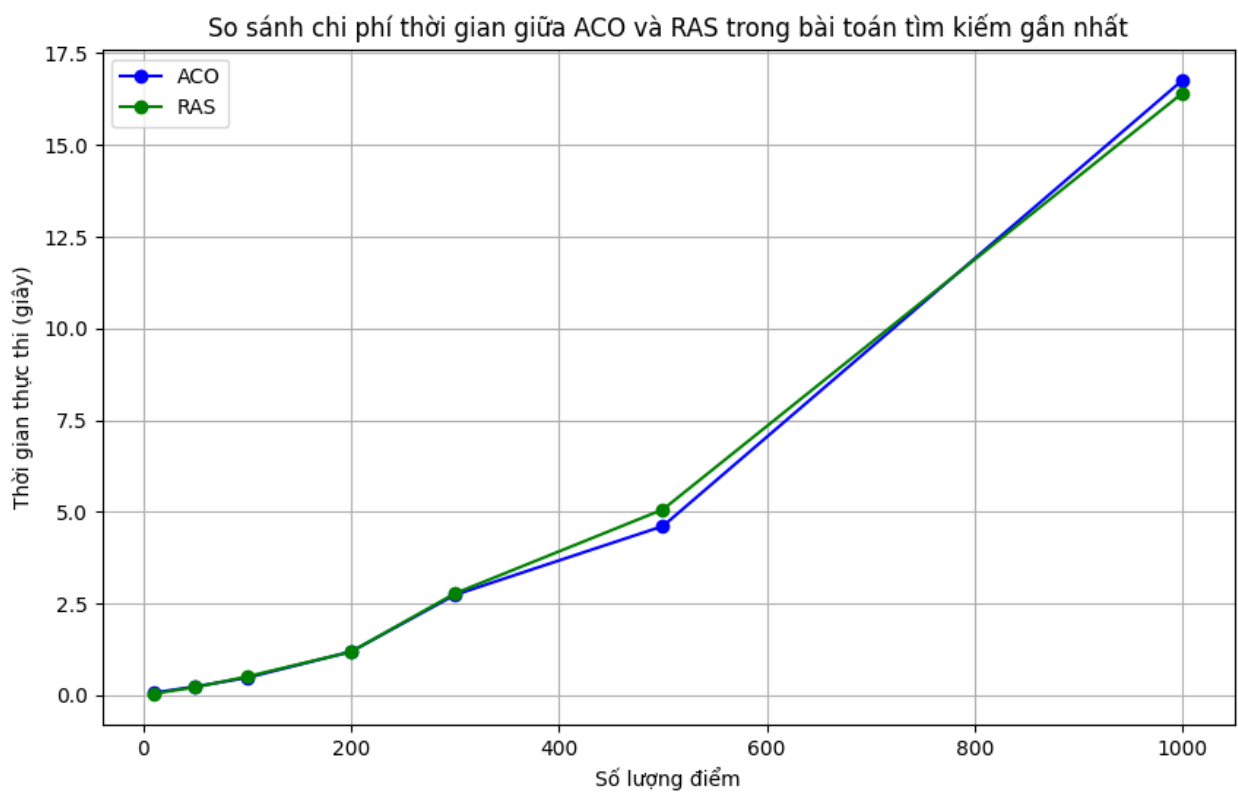
Tối ưu hơn như thế nào:

- + Cập nhật pheromone dựa trên xếp hạng: Các kiến được xếp hạng và chỉ những kiến có giả pháp tốt nhất mới được cập nhật pheromone nhiều hơn.

+ Giảm phụ thuộc vào kiến tồi: Kiến có hành trình kém không làm ảnh hưởng lớn đến pheromone, giúp hệ thống tránh bị ảnh hưởng bởi các giải pháp tồi.

+ Khác thác giải pháp tốt hơn: Tập trung vào khai thác các giải pháp tốt hơn trong quá trình tìm kiếm, giúp tăng tốc độ hội tụ và cải thiện hiệu quả.

RAS cải thiện hiệu quả bằng cách tập trung vào các kiến có giải pháp tốt nhất, tăng khả năng hội tụ sớm và giảm chi phí tính toán. Để kiểm định, chúng ta tiếp tục đo lường chi phí trên cùng bài toán có kết quả như bên dưới:



Hình 3.12 Biểu đồ so sánh chi phí thời gian ACO và RAS

Dựa vào biểu đồ thấy ACO cũng thường xu hướng mất nhiều thời gian hơn khi số lượng điểm. Trong khi RAS với cơ chế xếp hạng và cập nhật pheromone dựa trên kiến tốt hơn giúp tối ưu hoá quá trình tìm kiếm và thường có thời gian thực thi ngắn hơn so với ACO.

#5 Hybrid ACO (Kết hợp ACO với các thuật toán khác)

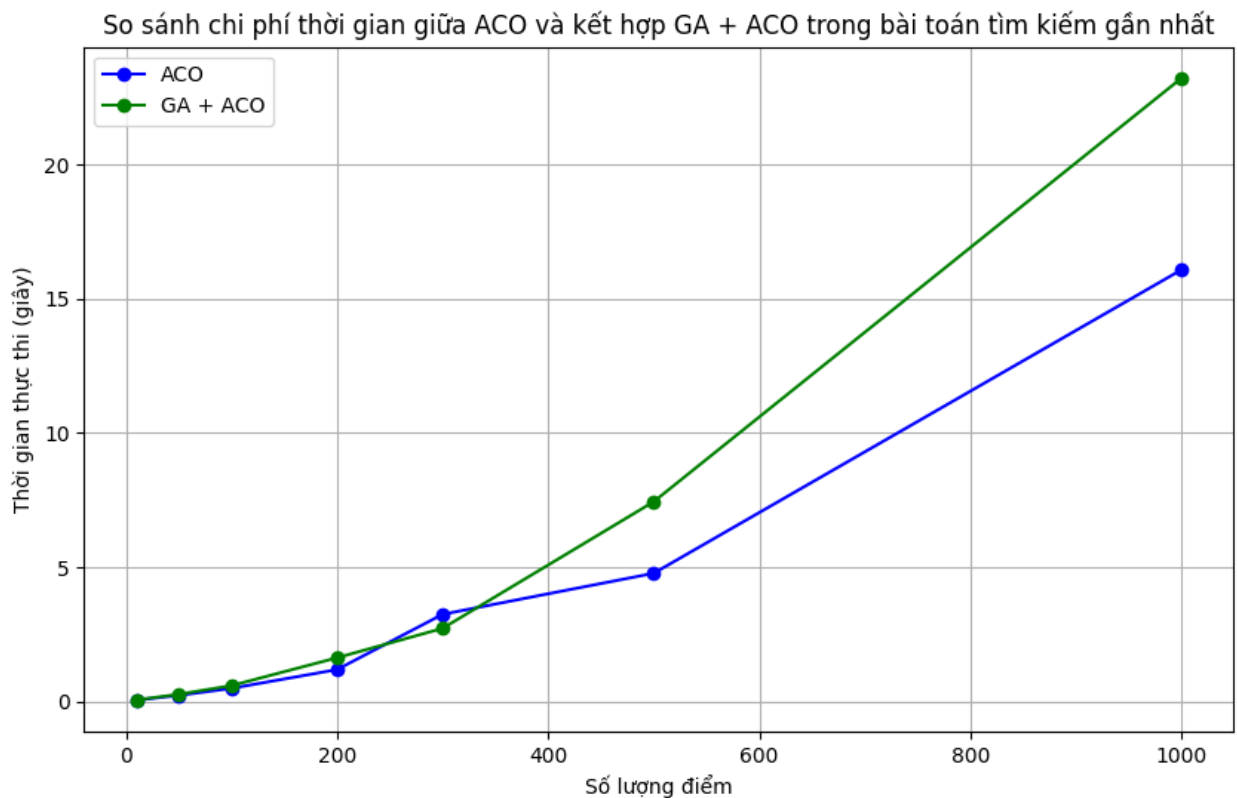
Một cách tiếp cận khác các biến thể còn lại là kết hợp ACO với các thuật toán tối ưu hoá khác như Genetic Algorithm (GA) hoặc Simulated Annealing (SA) có thể cải thiện khả năng tìm kiếm các giải pháp tốt hơn.

- Tối ưu hơn như thế nào:

+ Kết hợp GA vs ACO: Sử dụng GA để tạo ra 1 tập hợp các giải pháp tốt ban đầu tốt, sau đó áp dụng ACO để tinh chỉnh và tối ưu thêm.

+ Kết hợp ACO và Simulated Annealing: ACO có thể kết hợp với SA để tăng cường khả năng thoát khỏi bẫy cục bộ, bằng cách giảm dần pheromone trên các hành trình tồi.

Các phương pháp lai kết hợp sức mạnh của nhiều thuật toán giúp tối ưu hoá quá trình tìm kiếm, cho phép giải quyết các bài toán phức tạp hơn và tìm kiếm giải pháp tối ưu toàn cục tốt hơn. Cùng một phương pháp đo giữa biến thể lai GA + ACO và ACO gốc ta có kết quả như sau:



Hình 3.13 Biểu đồ so sánh chi phí thời gian ACO và GA+ACO

Theo biểu đồ ta thấy dễ dàng nhận định biến thể lai ACO + GA mang lại thời gian lâu hơn thuật toán gốc nhưng điều này không nhất thiết có nghĩa là GA + ACO không tốt hơn. Thời gian xử lý hơn có thể xuất phát từ việc GA thêm bước xử lý trước khi áp dụng ACO. Nhưng chúng ta cần xem xét thêm các yếu tố khác như chất lượng giải pháp và tốc độ hội tụ.

Một số điểm quan trọng cần xem xét:

- Chất lượng giải pháp:
 - + ACO có thể tốn ít thời gian hơn, nhưng nếu bị rơi vào giải pháp cục bộ không tốt, chất lượng giải pháp cuối cùng có thể kém hơn so với GA + ACO.
 - + GA+ACO: Sự kết hợp GA giúp tạo ra các giải pháp khởi tạo đa dạng hơn sau đó ACO sẽ tinh chỉnh các giải pháp này. Mặc dù mất nhiều thời gian hơn, nhưng GA + ACO thường dẫn đến giải pháp chất lượng hơn, đặc biệt trong những bài toán phức tạp.
- Tốc độ hội tụ:
 - + ACO có thể nhanh hơn, nhưng nó có thể gặp phải hiện tượng hội tụ quá sớm (premature convergence), nghĩa là nó sẽ dừng ở một giải pháp chưa tối ưu do các kiến tập trung vào các đường đi kém.
 - + GA + ACO: GA có khả năng khám phá không gian giải pháp tốt hơn trước khi ACO thực hiện tối ưu hoá. Nhờ vậy GA + ACO có thể tránh được hiện tượng hội tụ sớm và có khả năng tìm ra giải pháp tối ưu hơn.
- Phạm vi bài toán:
 - + ACO có thể hoạt động tốt hơn cho các bài toán nhỏ và không phức tạp vì không có sự thêm vào của GA.
 - + GA + ACO phù hợp hơn cho các bài toán lớn và phức tạp, nơi mà giải pháp cục bộ có thể dễ mắc phải và cần sự đa dạng trong các giải pháp khởi tạo.

3.5 Tổng kết:

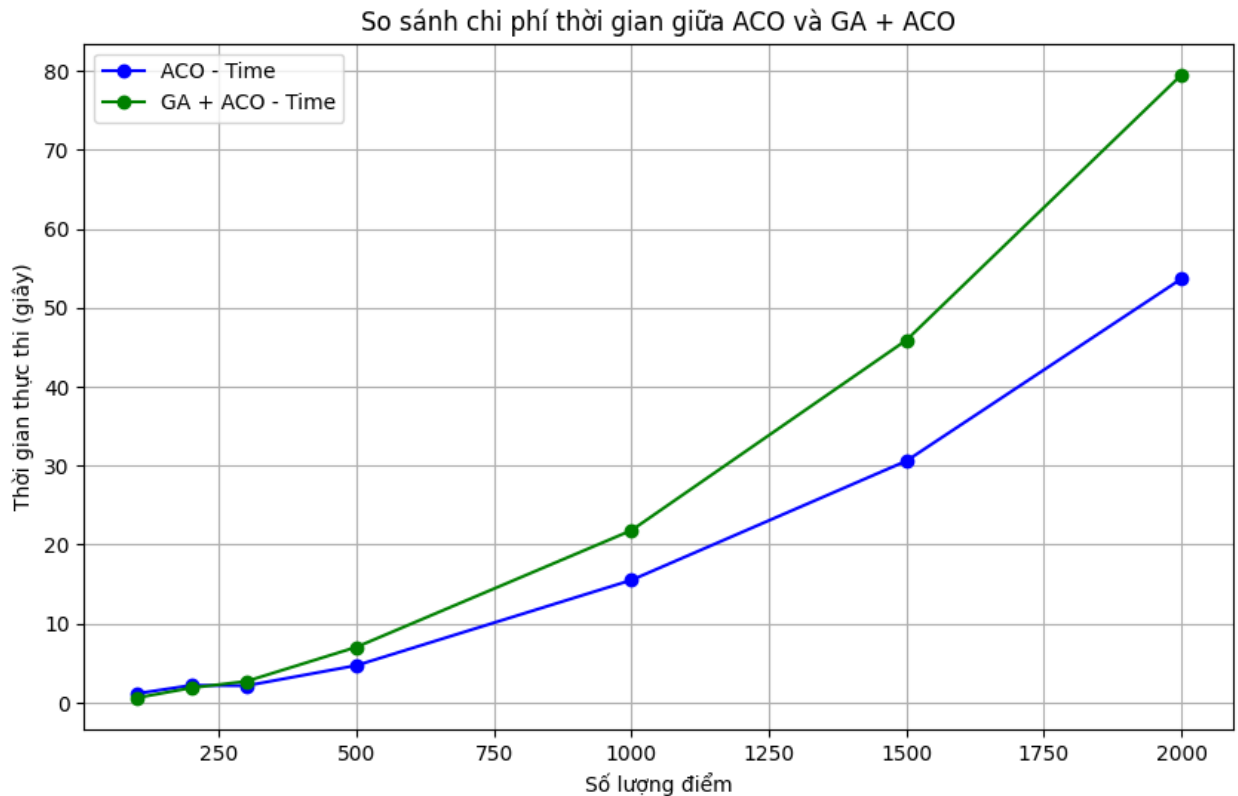
+ Nếu ưu tiên là thời gian và làm việc với bài toán không quá phức tạp, ACO gốc có thể là lựa chọn tốt hơn.

+ Nếu bạn cần chất lượng giải pháp cao hơn, đặc biệt khi làm việc với các bài toán phức tạp hoặc cần tránh hội tụ sớm, sự kết hợp GA + ACO. Có thể mang lại giải pháp tốt hơn mặc dù tốn thời gian hơn.

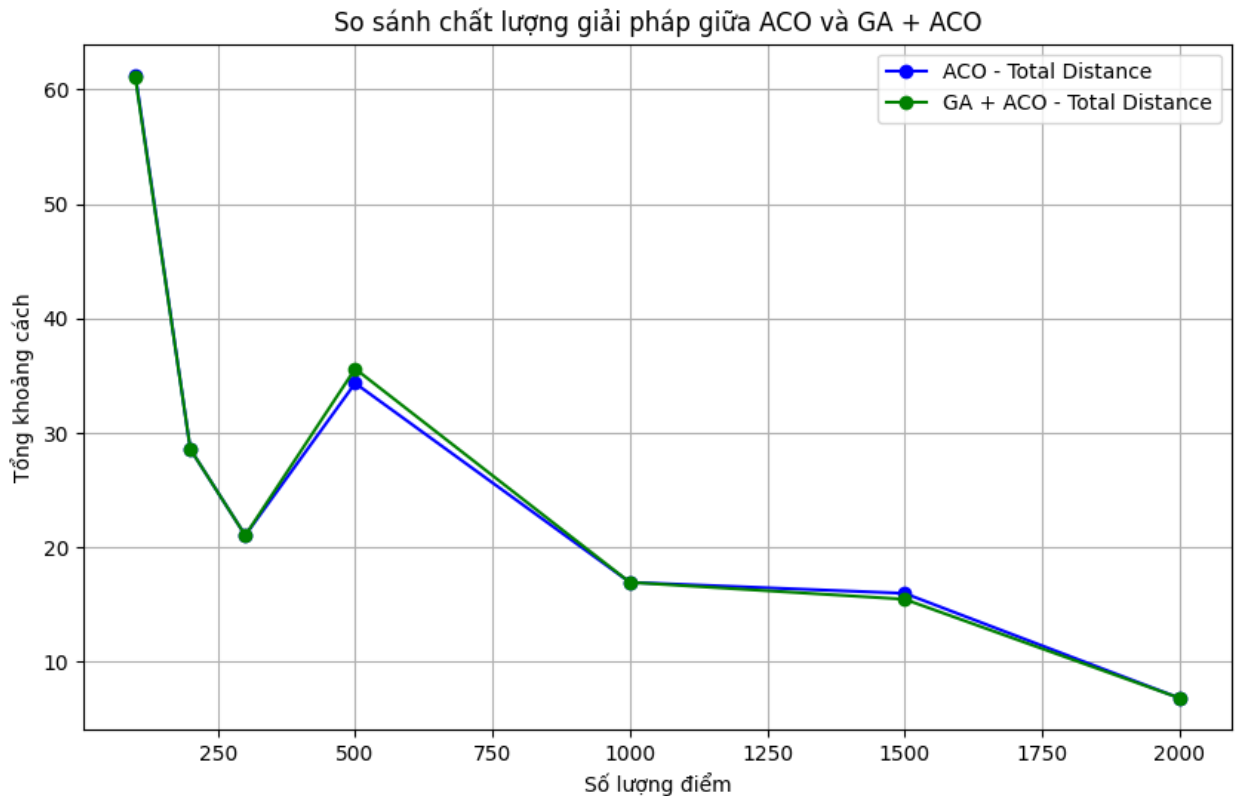
CHƯƠNG 4 : KẾT QUẢ VÀ THẢO LUẬN

4.1 Trình bày kết quả:

Với các kết quả đo chi phí & hiệu quả của các biến thể ACO áp dụng trong bài toán tìm kiếm, nhóm đặc biệt chú ý nhóm biến thể lai cụ thể GA + ACO do mang lại hiệu quả tốt ở tập mẫu lớn ở các mốc 1000, 2000, 5000, 7000, 10000 điểm. Cụ thể ở 2 biểu đồ về 2 tiêu chí thời gian & khoảng cách



Hình 4.1 Biểu đồ so sánh chi phí thời gian ACO và GA + ACO



Hình 4.2 Biểu đồ so sánh chất lượng giải pháp ACO và GA + ACO

Bằng cách gia. Tăng số lượng điểm và so sánh các tiêu chí khác ngoài thời gian xử lý cụ thể là tổng khoảng cách giữa các điểm trong hành trình gần nhất. Một lần đánh giá hiệu quả cao của nhóm biến thể GA + ACO so với thuật toán gốc

4.2 Phân tích kết quả:

- Mặc dù GA + ACO có thể mất nhiều thời gian hơn nhưng biểu đồ tổng khoảng cách chứng minh rằng GA + ACO có thể tìm ra các giải pháp tốt hơn (khoảng cách ngắn hơn), đặc biệt khi số lượng điểm lớn.
- ACO gốc có thể tốn ít thời gian hơn, nhưng giải pháp tìm được không tối ưu bằng so với GA + ACO.

CHƯƠNG 5 : KẾT LUẬN VÀ ĐỀ NGHỊ

5.1 Kết luận

Ở những tập mẫu lớn khoảng 10,000 điểm, các biến thể ACO sẽ có những ưu điểm khác nhau, và việc lựa chọn biến thể phù hợp phụ thuộc vào mục tiêu cụ thể thời gian thực thi hay chất lượng giải pháp. Ở mỗi biến thể nói riêng hay thuật toán nói chung không hoàn toàn mỹ cũng có nhược điểm tùy thuộc vào ngữ cảnh áp dụng phù hợp bài toán. Dưới đây ưu điểm / nhược điểm của các biến thể ACO khi áp dụng bài toán tìm kiếm gần nhất.

Thuật toán	Ưu điểm	Nhược điểm
ACS	<ul style="list-style-type: none">+ Khai thác mạnh: ACS tập trung vào hành trình tốt nhất và có thể hội tụ nhanh hơn nhờ cơ chế cập nhật pheromone cục bộ và toàn cục.+ Tối ưu thời gian: Cơ chế cập nhật pheromone cục bộ giúp quá trình tìm kiếm bớt phụ thuộc vào những giải pháp kém, do đó thời gian hội tụ sẽ nhanh hơn.	ACS có thể bị hội tụ quá sớm và bỏ qua các giải pháp tiềm năng khác khi bài toán có kích thước quá lớn.
MMAS	<ul style="list-style-type: none">+ Khám phá rộng: MMAS cho phép kiểm soát chặt chẽ giá trị pheromone, tránh trường hợp pheromone tập trung quá mức vào một vài hành trình.+ Tránh hội tụ sớm: nhờ giới hạn giá trị pheromone giữa T_{min} và T_{max}, MMAS tránh	Mặc dù MMAS có khả năng khám phá tốt hơn, nhưng nó có thể cần nhiều thời gian để hội tụ so với ACS.

	<p>được việc hệ thống hội tụ sớm vào giải pháp kém.</p> <p>+ Khả năng tìm kiếm giải pháp tốt hơn trong không gian rộng lớn, đặc biệt hữu ích khi bài toán có nhiều điểm trong trường hợp tập mẫu $> 10,000$ điểm.</p>	
RAS	<p>+ Cập nhật pheromone thông minh: RAS ưu tiên các kiến tốt hơn trong quá trình cập nhật pheromone, giúp tăng khả năng tìm kiếm các giải pháp tốt nhất.</p> <p>+ Giảm bớt ảnh hưởng của các giải pháp kém.</p>	Với bài toán lớn, quá trình xếp hạng có thể làm tăng chi phí tính toán, dẫn đến thời gian xử lý tăng lên đáng kể.
EAS	<p>+ Tập trung khai thác giải pháp tốt nhất: Pheromone được tăng cường mạnh mẽ cho hành trình của kiến ưu tú, giúp đẩy nhanh quá trình hội tụ về giải pháp tốt.</p> <p>+ Phù hợp cho bài toán lớn nếu giải pháp tốt cần được khai thác nhanh.</p>	EAS có thể dẫn đến hội tụ quá sớm, đặc biệt khi làm việc với bài toán phức tạp và có không gian giải pháp rộng như $10,000+$ điểm
GA + ACO	<p>+ Khám phá & khai thác tốt: GA giúp tìm kiếm các giải pháp khởi tạo tốt hơn bằng cách lai ghép và đột biến, sau đó ACO</p>	Kết hợp GA và ACO làm tăng thời gian xử lý, điều này có thể không phù hợp nếu bạn ưu tiên tốc độ hơn chất lượng giải pháp.

	<p>sẽ tinh chỉnh các giải pháp đó để tối ưu cục bộ.</p> <p>+ Tránh hội tụ sớm: Nhờ GA, thuật toán có thể tránh rơi vào các bẫy cục bộ và tìm được các giải pháp toàn cục tốt hơn.</p>	
--	---	--

5.2 Đề nghị

Cũng tùy theo yêu cầu bài toán, nhưng với phạm vi bài toán lớn như hơn 10,000 – 500,000 điểm thực tế, bạn cần một thuật toán có khả năng khám phá tốt và tránh hội tụ sớm, đồng thời đảm bảo tối ưu cục bộ mạnh mẽ. Dưới đây là các đề xuất:

- Nếu ưu tiên tốc độ và hội tụ nhanh:

+ ACS là lựa chọn tốt nhất vì nó tập trung mạnh vào khai thác các giải pháp tốt nhất và có cơ chế cập nhật pheromone giúp hội tụ nhanh hơn.

- Nếu ưu tiên chất lượng giải pháp:

+ MMAS là lựa chọn tốt hơn, đặc biệt khi bài toán có không gian giải pháp rộng và dễ bị hội tụ sớm. MMAS kiểm soát tốt quá trình khai thác và khám phá, giúp tìm ra các giải pháp toàn cục tốt hơn trong trường hợp có nhiều điểm.

- Nếu bạn cần giải pháp toàn cục tốt nhất và sẵn sàng chấp nhận thời gian xử lý lâu hơn:

+ Kết hợp GA + ACO có thể là lựa chọn tối ưu. GA sẽ giúp tránh các bẫy cục bộ và tạo ra các giải pháp tốt từ đầu, trong khi ACO tinh chỉnh thêm.

PHỤ LỤC

Link Source code: <https://github.com/tuanhhoang0710/ntt-aco-nhom4-2400000006-2400000053>

DANH MỤC TÀI LIỆU THAM KHẢO

- [1] TS. Cao Văn Kiên, Slide bài giảng Học máy và ứng dụng, Khoa Công nghệ thông tin – Đại học Nguyễn Tất Thành
- [2] Lê-Thị-Ngọc-Vân, 2.-Lê-Thị-Ngọc-Vân-46, Đại học Nguyễn Tất Thành
- [3] Trần Diệu Linh, Vũ Thị Mai Chi, Lê Khả Phiêu, Đỗ Huy Hoàng, Ngô Linh Chi (2023), Nghiên cứu KHSV 2023, Đại học Nguyễn Tất Thành
- [4] Quoc-Nam Tang, Tuan-Trung Tran (2016), 2016.VCM-Thuattoankien, Đại học Nguyễn Tất Thành
- [5] Bernhard Korte Jens Vygen, Combinatorial Optimization, Algorithms and Combinatorics
- [6] Vangelis Th.Paschos, Concepts of Combinatorial Optimization, Wiley
- [7] Vangelis Th.Paschos, Applications of Combinatorial Optimization, Wiley
- [8] TS Nguyễn Văn Long (2006), Phương pháp tối ưu, NXB Giao thông vận tải