ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

BÁO CÁO CUỐI KÌ

So sánh hiệu quả các thuật toán tối ưu trong bài toán TSP (Traveling Salesman Problem)

Nhóm thực hiện: Nhóm 3

DESIGN AND ANALYSIS ALGORITHM

Hà Nội - 05/2025

Báo Cáo Cuối Kỳ

Thiết kế và đánh giá thuật toán

Giảng viên hướng dẫn: PGS.TS. Nguyễn Thị Hồng Minh

Sinh viên thực hiện: Nguyễn Thành Trung - 22001672

Nguyễn Thị Ánh - 22000070

Nguyễn Tiến Đạt - 22000081

Nguyễn Khánh Đô - 22000083

Ngày 10 tháng 05 năm 2025

Tóm tắt nội dung

Bài toán Người Du Lịch (TSP - Traveling Salesman Problem) là một trong những bài toán tối ưu tổ hợp nổi bật, với mục tiêu tìm ra lộ trình ngắn nhất mà một người du lịch cần phải đi qua tất cả các thành phố, chỉ một lần, và quay lại điểm xuất phát. Tuy nhiên, đây là bài toán thuộc loại NP-khó, điều đó có nghĩa là không thể tìm ra giải pháp hiệu quả trong thời gian đa thức khi kích thước bài toán trở nên quá lớn, đẩy chúng ta vào một thế giới của sự phức tạp và thử thách.

Mục tiêu của nghiên cứu này là so sánh và đánh giá hiệu quả của bốn thuật toán tiêu biểu trong việc giải quyết bài toán TSP: ACO (Ant Colony Optimization), PSO (Particle Swarm Optimization), Thuật toán tham lam (Greedy Algorithm), và Thuật toán nhánh cận (Branch and Bound). Mỗi thuật toán mang trong mình một phương thức tiếp cận khác nhau, với những ưu điểm và nhược điểm riêng, từ đó mở ra những con đường giải quyết bài toán đầy sáng tạo.

Thuật toán ACO mô phỏng hành vi tinh vi của một đàn kiến trong tự nhiên, nơi pheromone được sử dụng như một chỉ dẫn giúp các con kiến tìm ra lộ trình tối ưu. Còn PSO lại lấy cảm hứng từ sự phối hợp tuyệt vời của đàn chim hay đàn cá, trong đó mỗi cá thể tìm kiếm giải pháp tối ưu nhờ vào sự tương tác qua lại giữa các cá thể trong bầy đàn. Đối với Thuật toán tham lam, phương pháp đơn giản nhưng mạnh mẽ, thuật toán luôn chọn lựa quyết định tối ưu trong từng bước mà không phải lo lắng về các bước tiếp theo, với hy vọng sẽ tìm ra giải pháp tốt nhất trong thời gian ngắn. Cuối cùng, Thuật toán nhánh cận phân chia không gian tìm kiếm thành các nhánh nhỏ hơn, từ đó loại bỏ những nhánh không khả thi, giúp ta nhanh chóng khoanh vùng các giải pháp tiềm năng. Nghiên cứu này sẽ tiến hành phân tích và so sánh các thuật toán trên nhiều phương diện, bao gồm độ chính xác, tốc độ hội tụ và khả năng giải quyết bài toán trong những bài toán có số lượng thành phố lớn, từ đó rút ra những nhận định quý giá về sự hiệu quả của từng thuật toán trong việc tối ưu hóa lộ trình cho bài toán TSP.

Mục lục

Chương 1

Giới Thiệu

Bài toán Người Du Lịch (TSP - Traveling Salesman Problem) là một trong những bài toán nổi bật trong lĩnh vực tối ưu tổ hợp, với mục tiêu tìm ra lộ trình ngắn nhất mà một người du lịch cần đi qua tất cả các thành phố một lần và quay lại điểm xuất phát. Bài toán này có thể được mô hình hóa dưới dạng đồ thị, trong đó các thành phố là các đỉnh và các con đường nối các thành phố là các cạnh, với trọng số của mỗi cạnh tương ứng với khoảng cách giữa các thành phố. Nếu giữa hai thành phố không có đường đi trực tiếp, trọng số của cạnh này sẽ được coi là một giá trị rất lớn. Khi đó, bài toán TSP trở thành bài toán tìm một chu trình Hamilton ngắn nhất, tức là tìm ra lộ trình khép kín sao cho tổng quãng đường đi qua tất cả các thành phố là ngắn nhất. TSP là bài toán thuộc loại **NP-khó**, có nghĩa là không có thuật toán hiệu quả nào có thể giải quyết bài toán này trong thời gian đa thức với kích thước đầu vào lớn. Mặc dù với bài toán cỡ nhỏ, ta có thể giải quyết bằng phương pháp tìm kiếm vét cạn để tìm ra giải pháp tối ưu, nhưng khi kích thước bài toán tăng lên, các phương pháp này không thể đáp ứng được yêu cầu về thời gian tính toán hợp lý. Chính vì vậy, việc tìm kiếm các giải pháp gần đúng hoặc đủ tốt trở thành một nhu cầu thiết yếu trong các ứng dụng thực tế. Các thuật toán tối ưu tổ hợp truyền thống đòi hỏi phải chứng minh tính hội tụ và ước lượng được tỷ lệ tối ưu so với giải pháp lý tưởng, điều này thường làm hạn chế số lượng thuật toán được công bố và không đáp ứng được yêu cầu đa dạng trong nghiên cứu và ứng dụng. Để khắc phục vấn đề này, các phương pháp tối ưu mềm ra đời, trong đó các thuật toán không nhất thiết phải tìm ra giải pháp tối ưu tuyệt đối, mà chỉ cần đưa ra những giải pháp có chất lượng đủ tốt trong thời gian tính toán hợp lý. Mục tiêu của nghiên cứu này là so sánh hiệu quả của bốn thuật toán nổi bật trong việc giải quyết bài toán TSP: **ACO (Ant Colony Optimization)**, **PSO (Particle Swarm Optimization)**, **Thuật toán tham**

lam (Greedy Algorithm) và **Thuật toán nhánh cận (Branch and Bound)**. Mỗi thuật toán sẽ được đánh giá dựa trên các yếu tố như độ chính xác, tốc độ hội tụ và khả năng giải quyết bài toán với số lượng thành phố lớn.

- **ACO (Ant Colony Optimization)**: Mô phỏng hành vi của đàn kiến trong việc tìm kiếm thức ăn. Thuật toán này sử dụng cơ chế pheromone để dẫn dắt các con kiến tìm ra lộ trình tối ưu, từ đó áp dụng vào bài toán TSP để tìm ra lộ trình ngắn nhất trong không gian tìm kiếm rộng lớn.
- **PSO (Particle Swarm Optimization)**: Dựa trên nguyên lý hoạt động của đàn chim hoặc cá, trong đó mỗi cá thể (particle) trong đàn tìm kiếm giải pháp tối ưu dựa trên sự tương tác của chính nó và các cá thể khác. Thuật toán này có khả năng hội tụ nhanh chóng và thích hợp cho việc giải quyết bài toán TSP.
- **Thuật toán tham lam (Greedy Algorithm)**: Là một phương pháp đơn giản và dễ triển khai, trong đó thuật toán chọn lựa quyết định tốt nhất tại mỗi bước mà không xét đến các bước tiếp theo. Mặc dù không đảm bảo giải pháp tối ưu toàn cục, nhưng thuật toán tham lam có thể mang lại kết quả nhanh chóng trong các bài toán với số lượng thành phố nhỏ.
- **Thuật toán nhánh cận (Branch and Bound)**: Thuật toán này phân chia không gian tìm kiếm thành các nhánh và loại bỏ các nhánh không khả thi, giúp giảm thiểu thời gian tính toán trong việc tìm kiếm giải pháp tối ưu.

Mục tiêu của nghiên cứu này là phân tích và so sánh hiệu quả của các thuật toán trên đối với bài toán TSP, từ đó đưa ra cái nhìn tổng quan về ưu và nhược điểm của từng thuật toán khi áp dụng vào các bài toán NP-khó với số lượng thành phố lớn.

Chương 2

Cơ sở lý thuyết thuật toán

2.1 Thuật toán Đàn K iến ACO:

2.1.1 Từ những con kiến trong tự nhiên tới thuật toán ACO.

Thuật toán ACO lấy ý tưởng từ việc kiếm thức ăn của đàn kiến ngoài thực tế để giải quyết các bài toán tối ưu tổ hợp. Chúng dựa trên cơ sở một đàn kiến nhân tạo, chúng được tính toán tìm kiếm thức ăn nhờ mùi lạ nhân tạo.

Cấu trúc cơ bản của thuật toán ACO: trong mỗi thuật toán, tất cả kiến đi xây dựng cách giải quyết bài toán bằng cách xây dựng một đồ thị. Mỗi cạnh của đồ thị miêu tả các bước kiến có thể đi được kết hợp từ hai loại thông tin hướng dẫn kiến di chuyển:

Thông tin kinh nghiệm giới hạn kinh nghiệm ưu tiên di chuyển từ nút r tới s trên cạnh a_{rs}. Nó được biểu diễn bằng ký hiệu η_{rs}. Thông tin này không được thay đổi bởi kiến trong suốt quá trình chạy thuật toán.

Thông tin mùi lạ nhân tạo giới hạn "nghiên cứu sự thèm muốn"của chuyển động của kiến nhân tạo và bắt chước mùi lạ thực tế của đàn kiến tự nhiên. Thông tin này bị thay đổi trong suốt quá trình thuật toán chạy, phụ thuộc vào cách giải quyết được tìm thấy bởi những con kiến. Nó được biểu diễn bằng ký hiệu τ_{rs}.

Giới thiệu các bước ảnh hưởng từ những con kiến thật vào ACO. Có hai vấn đề cần chú ý:

- Chúng trừu tượng hoá vài mô hình thức ăn của kiến ngoài thực tế để tìm ra đường đi tìm kiếm thức ăn ngắn nhất.
- Chúng bao gồm vài đặc điểm không giống với tự nhiên nhưng lại cho phép thuật toán phát triển chứa đựng cách giải quyết tốt tới bài toán bị cản (ví dụ: sử dụng

thông tin kinh nghiệm để hướng dẫn chuyển động của kiến).

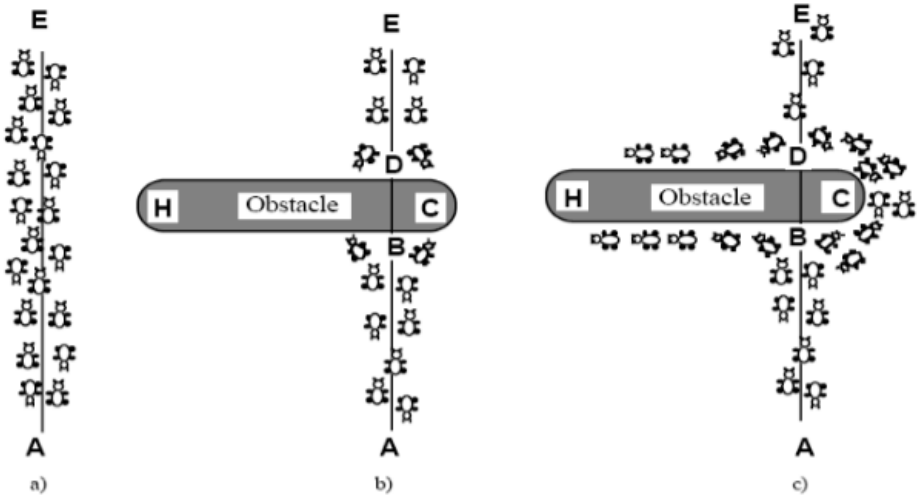
Cách thức hoạt động cơ bản của một thuật toán ACO như sau: m kiến nhân tạo di chuyển, đồng thời và không đồng bộ, qua các trạng thái liền kề của bài toán. Sự di chuyển này theo một tập quy tắc làm cơ sở từ những vùng thông tin có sẵn ở các thành phần (các nút). Vùng thông tin này bao gồm thông tin kinh nghiệm và thông tin mùi lạ để hướng dẫn tìm kiếm. Qua sự di chuyển trên đồ thị kiến xây dựng được cách giải quyết. Những con kiến sẽ giải phóng mùi lạ ở mỗi lần chúng đi qua một cạnh (kết nối) trong khi xây dựng cách giải quyết (cập nhật từng bước mùi lạ trực tuyến). Mỗi lần những con kiến sinh ra cách giải quyết, nó được đánh giá và nó có thể tạo luồng mùi lạ là hoạt động của chất lượng của cách giải quyết của kiến (cập nhật lại mùi lạ trực tuyến). Thông tin này sẽ hướng dẫn tìm kiếm cho những con kiến đi sau.

Hơn thế nữa, cách thức sinh hoạt động của thuật toán ACO bao gồm thêm hai thủ tục, sự bay hơi mùi lạ (pheromone trail evaporation) và hoạt động lạ (daemon actions). Sự bay hơi của mùi lạ được khởi sự từ môi trường và nó được sử dụng như là một kĩ thuật để tránh tìm kiếm bị dừng lại và cho phép kiến khảo sát vùng không gian mới. Daemon actions là những hoạt động tối ưu như một bản sao tự nhiên để thực hiện những nhiệm vụ từ một mục tiêu xa tới vùng của kiến.

2.1.2 Giới thiệu về thuật toán

Các thuật toán kiến là các thuật toán dựa vào sự quan sát các bầy kiến thực. Kiến là loại cá thể sống bầy đàn. Chúng giao tiếp với nhau thông qua mùi mà chúng để lại trên hành trình mà chúng đi qua. Mỗi kiến khi đi qua một đoạn đường sẽ để lại trên đoạn đó một chất mà chúng ta gọi là mùi. Số lượng mùi sẽ tăng lên khi có nhiều kiến cùng đi qua. Các con kiến khác sẽ tìm đường dựa vào mật độ mùi trên đường, mật độ mùi càng lớn thì chúng càng có xu hướng chọn. Dựa vào hành vi tìm kiếm này mà đàn kiến tìm được đường đi ngắn nhất từ tổ đến nguồn thức ăn và sau đó quay trở tổ của mình.

Sau đây là ví dụ về luồng đi của đàn kiến thực tế.



Hình 2.1: Luồng đi của đàn kiến thực tế

a) Kiến đi theo đường thẳng giữa A và E

b) Khi có chướng ngại vật kiến sẽ chọn hướng đi, có hai hướng với khả năng kiến sẽ chọn là như nhau.

c) Trên đường ngắn hơn thì nhiều mùi (pheromone) hơn.

Thuật toán tối ưu bầy kiến (ACO) nghiên cứu các hệ thống nhân tạo dựa vào hành vi tìm kiếm của bầy kiến thực và được sử dụng để giải quyết các vấn đề về tối ưu rời rạc.Thuật toán bầy kiến siêu tìm kiếm (ACO meta-heuristic) lần đầu tiên được Dorigo, Di Caro và Gambardella đề xuất vào năm 1999.

Metaheuristic là một tập các khái niệm về thuật toán được sử dụng để xác định các phương thức tìm kiếm thích hợp cho một tập các vấn đề khác nhau. Hay nói cách khác, một siêu tìm kiếm (meta-heuristic) có thể coi là một phương thức tìm kiếm đa năng.

ACO là một meta-heuristic, trong đó một tập các con kiến nhân tạo phối hợp tìm kiếm các giải pháp tốt cho các vấn đề về tối ưu rời rạc. Sự phối hợp là yếu tố cối lõi của các thuật toán ACO. Các con kiến nhân tạo liên lạc với nhau thông qua trung gian mà ta thường gọi là mùi.

Các thuật toán ACO được sử dụng để giải quyết các vấn đề về tối ưu tổ hợp tĩnh và động. Các vấn đề tĩnh là các vấn đề mà ở đó các đặc tính của vấn đề là không thay đổi trong suốt quá trình giải quyết vấn đề. Còn các vấn đề động thì ngược lại là một hàm các tham số mà giá trị của nó là động hay thay đổi trong quá trình giải quyết vấn đề, ví dụ bài toán người đưa thư là một vấn đề dynamic problem.

Hệ thống ACO lần đầu tiên được Marco Dorigo giới thiệu trong luận văn của mình vào năm 1992, và được gọi là Hệ thống kiến (Ant System, hayAS). AS là kết quả của việc

nghiên cứu trên hướng tiếp cận trí tuệ máy tính nhằm tối ưu tổ hợp mà Dorigo được hướng dẫn ở Politecnico di milano với sự hợp tác của Alberto Colorni và Vittorio Maniezzo. AS ban đầu được áp dụng cho bài toán người du lịch (TSP) và QAP.

Cũng vào năm 1992, tại hội nghị sự sống nhân tạo lần đầu tiên ở châu Âu , Dorigo và các cộng sự đã công bố bài: sự tối ưu được phân bố bởi đàn kiến.

Tiếp theo tại hội nghị quốc tế thứ hai về giải quyết các vấn đề song song trong tự nhiên ở Hà Lan (1992), ông và các cộng sự đã công bố bài: nghiên cứu về các đặc tính của một giải thuật kiến.

Kể từ năm 1995 Dorigo, Gambardella và Stützle đã phát triển các sơ đồ AS khác nhau. Dorigo và Gambardella đã đề xuất Hệ thống bầy kiến (Ant Colony System, hay ACS) trong khi Stützle and Hoos đề xuất MAX-MIN Ant System (MMAS). Tất cả đều áp dụng cho bài toán người du lịch đối xứng hay không đối xứng và cho kết quả mỹ mãn. Dorigo, Gambardella and Stützle cũng đề xuất những phiên bản lai của ACO với tìm kiếm địa phương.

Vào năm 1995, L.M. Gambardella và M. Dorigo đã đề xuất hệ thống Ant-Q, là một cách tiếp cận học tăng cường cho cho bài toán TSP.Và nó được áp dụng trong Học Máy.

Tiếp đó, vào năm 1996, trong bài báo công nghệ của mình tại Bruxelles M. Dorigo và L.M. Gambardella đã công bố hệ thống Ant Conoly System. Đây là hệ thống đề cập đến cách học phối hợp áp dụng cho bài toán TSP.

Cũng trong năm 1996 này, T. Stützle và H. H. Hoos đã đề xuất hệt thống Max-Min Ant System . Đây là một hệ thống cải tiến hệ thống AntSystem ban đầu và được đánh giá là hệ thống tính toán trong tương lai.

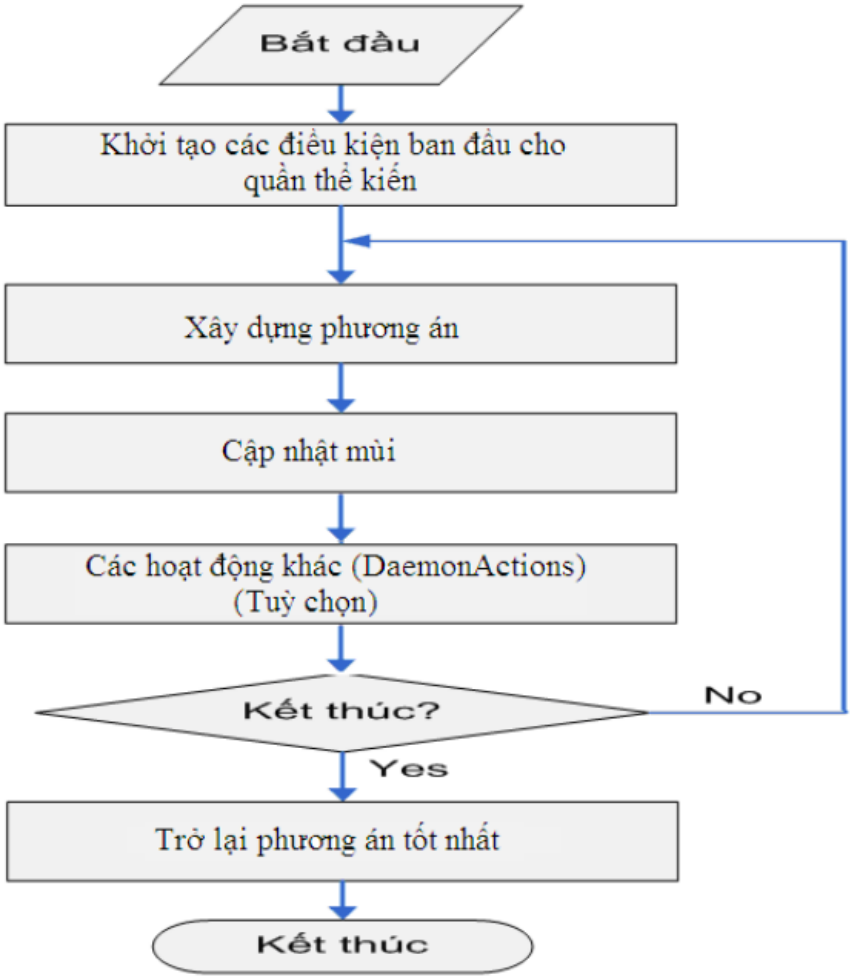
Sau đó, vào năm 1997, G. Di Caro và M. Dorigo đã đề xuất hệ thống AntNet. Đây là cách tiếp cận về định hướng sự thích nghi. Và phiên bản cuối cùng của hệ thống AntNet về điều khiển mạng truyền thông đã được công bố vào năm 1998.

Cũng trong năm 1997, hệ thống Rank-based Ant System, một hệ thống cải tiến hệ thống kiến ban đầu về nghiên cứu hệ thống tính toán đã được đề xuất bởi B. Bullnheimer, R. F. Hartl và C. Strauss. Phiên bản cuối cùng của hệ thống này được công bố vào năm 1999.

Vào năm 2001, C. Blum, A. Roli, và M. Dorigo đã cho công bố về hệ thống kiến mới là Hyper Cube – ACO. Phiên bản mở rộng tiếp đó đã được công bố vào năm 2004.

Hầu hết các nghiên cứu gần đây về ACO tập trung vào việc phát triển các thuật toán biến thể để làm tăng hiệu năng tính toán của thuật toán Ant System ban đầu.

2.1.3 Sơ đồ chung thuật toán đàn kiến



Hình 2.2: Sơ đồ chung của thuật toán đàn kiến

2.1.4 Các bước giải quyết bài toán đàn kiến

Từ thuật toán trên ta có thể rút ra các bước giải quyết một bài toán ứng dụng với thuật toán đàn kiến:

Bước 1: Thể hiện bài toán

Bài toán được biểu diễn trong khung của tập các thành phần và sự chuyển đổi hoặc bởi một đồ thị được đánh dấu, tạo nền tảng để kiến xây dựng cách giải quyết.

Bước 2: Định nghĩa mùi lạ τ_{rs}

Mùi lạ τ_{rs} là một xu hướng quyết định quan trọng trong thuật toán. Đây là bước chủ yếu trong việc hình thành thuật toán ACO. Việc định nghĩa mùi lạ không phải là một nhiệm

vụ tầm thường, vì nó yêu cầu sự tính toán bên trong bài toán và có liên quan trực tiếp đến đáp án của bài toán.

Bước 3: Định nghĩa thông tin kinh nghiệm η_{rs}

Mỗi quyết định cần được định nghĩa thông tin kinh nghiệm η_{rs}, kết hợp với mỗi thành phần hoặc trạng thái chuyển đổi. Thông tin kinh nghiệm là yếu tố chủ chốt trong việc tìm kiếm lời giải, đặc biệt trong các vùng mà thuật toán gặp khó khăn hoặc không thể ứng dụng.

Bước 4: Tạo vùng tìm kiếm hiệu quả

Nếu có thể, cần tạo ra một vùng tìm kiếm hiệu quả cho bài toán sau đáp án. Nhiều ứng dụng của ACO trên các bài toán tối ưu tổ hợp NP-hard đã chứng minh rằng kết quả tốt nhất đạt được khi thuật toán ACO có vùng tìm kiếm hợp lý và lạc quan.

Bước 5: Lựa chọn thuật toán ACO

Lựa chọn một thuật toán ACO phù hợp và ứng dụng nó vào bài toán cần giải quyết.

Bước 6: Tối ưu hóa tham số của thuật toán

Việc tối ưu hóa tham số là rất quan trọng để đạt hiệu quả cao nhất. Một điểm bắt đầu hợp lý là sử dụng các giá trị tham số mặc định đã được chứng minh hiệu quả trên các bài toán đơn giản. Ngoài ra, thủ tục động để điều chỉnh tham số cũng có thể được áp dụng nhằm nâng cao hiệu suất của thuật toán. Các bước trên cung cấp hướng dẫn cụ thể trong việc xây dựng và áp dụng thuật toán ACO để giải quyết các bài toán tối ưu tổ hợp. Việc thực hiện đúng từng bước sẽ đảm bảo thuật toán đạt hiệu quả tối đa trên các bài toán thực tế.

2.1.5 Các công thức

Xác suất để một kiến chọn thành phố tiếp theo được biểu diễn như sau:

$$p_{ij}^k(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^\alpha \times [\eta_{ij}(t)]^\beta}{\sum\limits_{s \in allow_k} [\tau_{is}(t)]^\alpha \times [\eta_{is}(t)]^\beta}, & j \in allow_k \\ 0, & j \notin allow_k \end{cases}$$

Trong đó:

- $allow_k$: Tập hợp các thành phố mà kiến k có thể chọn tiếp theo.
- $\eta_{ij}(t)$: Hàm kỳ vọng thể hiện mức độ mong muốn của kiến khi di chuyển từ thành phố i đến thành phố j.

  $\eta_{ij}(t)$ có thể tính theo nghịch đảo của khoảng cách giữa hai thành phố: $\eta_{ij}(t) = \frac{1}{d_{ij}}$.
- $\tau_{ij}(t)$: Mật độ pheromone trên cạnh ij tại thời điểm t.
- α và β: Các hệ số điều chỉnh ảnh hưởng của pheromone và mức độ kỳ vọng.

Quá trình bay hơi và tích lũy pheromone được mô tả bởi các công thức lặp sau:

$$\begin{cases} \tau_{ij}(t+1) = \rho \times \tau_{ij}(t) + \Delta\tau_{ij}, \quad 0 < \rho < 1 \\ \Delta\tau_{ij}(t, t+1) = \sum_{k=1}^{M} \Delta\tau_{ij}^{k}(t, t+1) \\ \Delta\tau_{ij}^{k}(t, t+1) = Q_t \times Z(N_c, k) \end{cases}$$

Trong đó:

- ρ: Hệ số pheromone còn lại, giúp điều chỉnh mức độ bay hơi của pheromone.
- $\Delta\tau_{ij}(t, t+1)$: Tổng lượng pheromone được tất cả các kiến phát hành trên cung ij trong vòng lặp từ t đến $t+1$.
- $\Delta\tau_{ij}^{k}(t, t+1)$: Lượng pheromone mới được kiến k phát hành trên cung ij.
- Q_t: Hệ số tăng cường pheromone, giúp điều chỉnh lượng pheromone mới phát hành.
- $Z(N_c, k)$: Hiệu quả của giải pháp mà kiến thứ k tìm được trong vòng lặp N_c.

2.1.6 Ưu điểm và nhược điểm

Ưu điểm của thuật toán ACO

1. **Khả năng tìm kiếm giải pháp gần tối ưu**: ACO có khả năng tìm kiếm giải pháp gần tối ưu cho các bài toán tối ưu tổ hợp phức tạp, đặc biệt là các bài toán như TSP (Traveling Salesman Problem), mà các phương pháp tối ưu cổ điển không thể giải quyết hiệu quả.

2. **Khả năng hội tụ tốt**: Thuật toán ACO có khả năng hội tụ dần dần đến giải pháp tối ưu trong quá trình lặp lại. Bằng cách điều chỉnh lượng pheromone và khai thác thông tin từ các "kiến"khác, thuật toán có thể cải thiện dần dần các giải pháp qua các vòng lặp.

3. **Khả năng giải quyết bài toán lớn**: ACO có khả năng giải quyết các bài toán lớn, phức tạp mà không bị phụ thuộc vào kích thước của không gian tìm kiếm quá nhiều. Dây là một ưu điểm lớn so với các thuật toán tìm kiếm vét cạn hay các phương pháp không hiệu quả với bài toán có kích thước lớn.

4. **Tính linh hoạt và có thể mở rộng**: ACO là thuật toán có tính linh hoạt cao, có thể áp dụng cho nhiều loại bài toán tối ưu khác nhau như bài toán TSP, bài toán phân phối, bài toán tối ưu hóa các mạng lưới, v.v. Hơn nữa, thuật toán có thể mở rộng để cải thiện hiệu suất hoặc để thích ứng với các yêu cầu đặc biệt của bài toán.

5. **Khả năng học hỏi từ các giải pháp trước đó**: Thuật toán ACO có khả năng học hỏi từ các giải pháp đã tìm được qua pheromone, giúp "kiến"có thể điều chỉnh chiến lược và cải thiện quá trình tìm kiếm trong các vòng lặp tiếp theo.

Nhược điểm của thuật toán ACO

1. **Chi phí tính toán cao**: Mặc dù ACO có khả năng tìm kiếm giải pháp tốt, nhưng chi phí tính toán của thuật toán có thể rất cao, đặc biệt là khi bài toán có kích thước lớn hoặc số lượng "kiến"quá lớn. Việc duy trì pheromone và cập nhật thông tin giữa các cá thể trong quá trình tìm kiếm có thể gây tốn tài nguyên tính toán.

2. **Khả năng hội tụ chậm**: ACO có thể cần nhiều vòng lặp để hội tụ đến giải pháp tối ưu. Trong một số trường hợp, thuật toán có thể hội tụ quá sớm vào các giải pháp không tối ưu nếu không có cơ chế điều chỉnh pheromone hợp lý, dẫn đến hiện tượng "local optima"(giải pháp tối ưu cục bộ).

3. **Cần tinh chỉnh tham số**: ACO yêu cầu phải điều chỉnh các tham số như tốc độ bay hơi của pheromone, số lượng kiến, và các yếu tố liên quan đến chiến lược tìm kiếm. Việc lựa chọn các giá trị tham số phù hợp là rất quan trọng và có thể mất nhiều thời gian thử nghiệm.

4. **Phụ thuộc vào cơ chế pheromone**: Thuật toán ACO chủ yếu dựa vào cơ chế pheromone để điều chỉnh chiến lược tìm kiếm, và việc này có thể gặp khó khăn

trong việc điều chỉnh pheromone sao cho không quá mạnh (dẫn đến hội tụ nhanh vào giải pháp không tối ưu) hoặc quá yếu (dẫn đến sự thiếu hụt trong quá trình khám phá không gian tìm kiếm).

5. **Vấn đề với bài toán đa mục tiêu**: Khi áp dụng ACO cho các bài toán tối ưu hóa đa mục tiêu, việc duy trì cân bằng giữa các mục tiêu có thể trở thành một vấn đề phức tạp, đòi hỏi các cải tiến hoặc thuật toán bổ sung để xử lý.

2.2 Thuật toán Tối ưu Bầy Đàn PSO

2.2.1 Giới thiệu

Hãy tưởng tượng việc giải các câu đố phức tạp bằng cách học từ những chuyển động phối hợp của chim và cá. Thuật toán Tối ưu Bầy Đàn (PSO) làm được điều đó. Thuật toán PSO là một cách thông minh để giải quyết các vấn đề khó bằng cách bắt chước cách các sinh vật làm việc cùng nhau. PSO sử dụng nhiều tác nhân nhỏ di chuyển xung quanh để tìm ra câu trả lời tốt nhất. Mỗi tác nhân nhớ giải pháp tốt nhất của chính nó và giải pháp tốt nhất từ những người hàng xóm của nó. Điều này giúp chúng làm việc cùng nhau và tìm ra câu trả lời tốt nhất nhanh hơn. Khi chúng ta khám phá PSO, chúng ta sẽ hiểu rõ hơn về cách mà sự hợp tác giữa các tác nhân ảo giúp chúng ta giải quyết các vấn đề thách thức trong nhiều lĩnh vực khác nhau.

2.2.2 Thuật toán Tối ưu Bầy Đàn (PSO)

PSO (Particle Swarm Optimization) là khung thuật toán chung dựa trên kinh nghiệm của bầy đàn được đề xuất bởi Kennedy và Eberhart. Nó là một khung thuật toán thông minh dựa trên bầy đàn, mô phỏng lại hành vi xã hội của bầy chim hay đàn cá khi đi tìm nguồn thức ăn.

Một cá thể (particle) được thể hiện trong PSO tương tự như một con chim hoặc một con cá tìm kiếm thức ăn trong không gian tìm kiếm của nó. Sự di chuyển của mỗi cá thể là sự kết hợp giữa vận tốc và hướng di chuyển. Vị trí của mỗi cá thể tại bất kỳ thời điểm nào cũng bị ảnh hưởng bởi vị trí tốt nhất của nó và vị trí tốt nhất của cả bầy đàn. Hiệu quả đạt được của một cá thể được xác định bởi một giá trị thích nghi, giá trị này được xác định phụ thuộc vào từng bài toán .

Trong PSO, quần thể bao gồm các cá thể trong không gian của bài toán. Các cá thể được

khởi tạo một cách ngẫu nhiên. Mỗi cá thể sẽ có một giá trị thích nghi, giá trị này được xác định bởi một hàm thích nghi để tối ưu trong mỗi thế hệ. Trong mỗi thế hệ, mỗi cá thể thay đổi vận tốc và thay đổi vị trí của nó theo thời gian. Dựa vào giá trị thích nghi, mỗi cá thể tìm ra giải pháp tối ưu của nhóm trong không gian tìm kiếm nhiều chiều. Sau đó, giải pháp tối ưu của nhóm sẽ được so sánh với giải pháp tối ưu toàn cục của cả bầy đàn để cập nhật lại giá trị cho giải pháp phần tử của nhóm và giải pháp tối ưu toàn cục để tìm ra giải pháp tối ưu nhất.

2.2.3 Mô hình

Để áp dụng khung thuật toán PSO vào bài toán cụ thể chúng ta phải xác định được vị trí, vận tốc, hàm thích nghi, vị trí tối ưu cực bộ của từng cá thể và vị trí tối ưu toàn cục của bầy đàn.

Chúng ta xét quần thể gồm Y cá thể, mỗi cá thể x lặp N lần để tìm kiếm thức ăn trong không gian M_x chiều. Như vậy, tại vòng lặp thứ i, $(i = 1...N)$ mỗi cá thể x, $(x = 1...Y)$ sẽ có M_x vị trí, M_x vận tốc và được biểu diễn:

$$P_{ix} = \{p_{ix}^1, p_{ix}^2, ..., p_{ix}^{M_x}\} \tag{2.1}$$

$$V_{ix} = \{v_{ix}^1, v_{ix}^2, ..., v_{ix}^{M_x}\} \tag{2.2}$$

Trong đó:

- p_{ix}^j là vị trí của cá thể x ở vòng lặp i tại chiều j, $(j = 1...M_x)$.
- v_{ix}^j là vận tốc của các thể x ở vòng lặp i tại chiều j, $(j = 1...M_x)$.

Trong bài toán người du lịch ta có Y thành phố, mỗi thành phố được nối với nhiều thành phố gần nhất. Mỗi cá thể x $(x = 1...Y)$ tìm kiếm thành phố trên không gian M_x chiều để tìm ra phương thức hợp lý tại lần lặp thứ i, mỗi cá thể x sẽ tìm ra các thành phố vào M_x chiều thành phố gần nhất theo từng thời gian. Giá trị của p_{ijx} là thành phố j lần gần nhất với cá thể x, và giá trị v_{ijx} là giá trị ngẫu nhiên từ -M_x tới M_x là số thành phố gần thành phố x.

Để đạt được mục tiêu là tổng chi phí đi qua các thành phố là nhỏ nhất, chúng ta xây dựng hàm thích nghi để cá thể x chọn ra thành phố lần cận tại lần lặp thứ i như sau:

$$F(p_{ix}^j) = \frac{1}{c_{ijx}} \tag{2.3}$$

Trong đó: c_{ijx} là trọng số của thành phố x với thành phố j tại lần lặp thứ i.

Khi trọng số c_{ijx} càng cao thì giá trị hàm thích nghi càng thấp. Ngược lại, khi trọng số c_{ijx} càng thấp thì giá trị hàm thích nghi càng cao nên xác suất để thành phố x chọn thành phố j càng lớn.

Từ hàm thích nghi ở công thức (2.3), ta tiến hành xây dựng công thức để tìm vị trí tối ưu cục bộ của cá thể x tại chiều $j+1$ như sau:

$$p_{ix}^{j+1} = \begin{cases} p_{ix}^{j+1} & \text{nếu } F(p_{ix}^{j+1}) \geq F(p_{ix}^{j}) \\ p_{ix}^{j} & \text{ngược lại} \end{cases} \tag{2.4}$$

Vị trí tối ưu cực bộ tại vị trí $j+1$ được cập nhật nếu giá trị của hàm thích nghi tại vị trí $j+1$ lớn hơn hoặc bằng giá trị của hàm thích nghi tại vị trí trước đó, ngược lại nó vẫn giữ lại giá trị của hàm thích nghi trước đó.

Sau khi xác định được vị trí tối ưu cục bộ của từng cá thể, ta tiến hành xác định vị trí tối ưu cục bộ lớn nhất của các cá thể (Pbest) và vị trí tối ưu toàn cục của cả đàn (Gbest) như sau:

Sau khi mỗi cá thể x ($x = 1...Y$) tìm kiếm được thành phố lần cận trên không gian M_x chiều, mỗi cá thể tìm được vị trí tối ưu nhất (Pbestx) bằng cách phân tích các vị trí tối ưu trên mỗi chiều và được xác định:

$$Pbest_x = \max_{x=1,...,Y, j=1,...,M_x} \{p_{ix}^{j}\} \tag{2.5}$$

Vị trí tối ưu toàn cục của cả đàn (Gbest) phụ thuộc vào vị trí tối ưu cục bộ lớn nhất của từng cá thể và được xác định như sau:

$$Gbest = \max\{Pbest_x\} \tag{2.6}$$

Mỗi cá thể dựa vào vận tốc hiện tại và khoảng cách từ $Pbest_x$ đến $Gbest$ để thay đổi vị trí và điều chỉnh tốc độ của nó như sau:

$$v_x^{j+1} = v_x^j + c_1 z_1 (Pbest_x - pos_x^j) + c_2 z_2 (Gbest - pos_x^j) \tag{2.7}$$

$$pos_x^{j+1} = pos_x^j + v_x^j \tag{2.8}$$

Trong đó:

- pos_x^j: vị trí hiện tại của cá thể x tại chiều j.

- c_1, c_2: hệ số gia tốc.
- z_1, z_2: là số ngẫu nhiên giữa 0 và 1.
- v_x^j: vận tốc của cá thể x tại chiều j.

2.2.4 Ưu điểm và nhược điểm

Ưu điểm của thuật toán PSO

Thuật toán tối ưu bầy đàn (PSO) đã thu hút sự chú ý đáng kể trong nhiều năm qua nhờ vào các ưu điểm vượt trội của nó. Dưới đây là những điểm mạnh chính của PSO:

1. **Đơn giản và dễ triển khai:** PSO có cấu trúc đơn giản và không yêu cầu tính toán phức tạp. Điều này giúp cho việc triển khai và áp dụng thuật toán trở nên dễ dàng hơn rất nhiều, ngay cả đối với những bài toán phức tạp.
2. **Khả năng hội tụ nhanh:** Đặc biệt là trong các bài toán có không gian tìm kiếm nhỏ hoặc dễ dàng tối ưu hóa. Với khả năng hội tụ nhanh, PSO có thể tìm ra nghiệm gần đúng trong thời gian ngắn, giúp tiết kiệm tài nguyên tính toán.
3. **Khả năng tìm kiếm toàn cục:** PSO có thể khám phá không gian tìm kiếm rộng và tìm ra các cực trị toàn cục, tránh việc mắc kẹt trong các cực trị cục bộ. Điều này khiến PSO trở thành một công cụ mạnh mẽ trong các bài toán tối ưu hóa phức tạp.
4. **Tính linh hoạt:** PSO có thể được áp dụng cho nhiều loại bài toán tối ưu hóa khác nhau, từ những bài toán đơn giản đến các bài toán phức tạp trong nhiều lĩnh vực, từ khoa học máy tính đến kỹ thuật và tài chính.

Những ưu điểm này khiến PSO trở thành một lựa chọn phổ biến trong việc giải quyết các vấn đề tối ưu hóa.

Nhược điểm của thuật toán PSO

Dù PSO có nhiều ưu điểm vượt trội, nhưng cũng tồn tại một số nhược điểm cần được khắc phục để tối ưu hóa hiệu quả sử dụng thuật toán:

1. **Khả năng hội tụ chậm ở không gian tìm kiếm lớn:** Với các bài toán có không gian tìm kiếm rộng, PSO có thể gặp khó khăn trong việc hội tụ nhanh chóng đến

nghiệm tối ưu. Điều này có thể làm giảm hiệu quả tính toán, đặc biệt khi không gian tìm kiếm quá lớn hoặc quá phức tạp.

2. **Khó điều chỉnh tham số:** Các tham số như w, c_1, c_2 ảnh hưởng lớn đến hiệu quả của thuật toán và việc điều chỉnh chúng không phải lúc nào cũng dễ dàng. Điều này có thể yêu cầu thử nghiệm nhiều lần để tìm ra cấu hình tham số tối ưu, gây tốn thời gian và tài nguyên.

3. **Mắc kẹt ở cực trị cục bộ:** Trong một số trường hợp, PSO có thể bị mắc kẹt ở các cực trị cục bộ nếu không có chiến lược thích hợp để tránh việc tìm kiếm bị giới hạn trong một vùng nhỏ. Điều này dẫn đến việc thuật toán không thể tìm ra cực trị toàn cục, làm giảm tính hiệu quả của quá trình tối ưu hóa.

Mặc dù PSO có thể gặp phải những vấn đề này, các phương pháp cải tiến và điều chỉnh chiến lược tìm kiếm có thể giúp giảm thiểu những nhược điểm trên.

2.3 Thuật toán tham lam

2.3.1 Giới thiệu

Giải thuật tham lam là một phương pháp tiếp cận phổ biến trong việc giải quyết các bài toán tối ưu hóa tổ hợp. Với chiến lược lựa chọn giải pháp tốt nhất tại mỗi bước đi, thuật toán hướng tới việc đạt được giải pháp tối ưu toàn cục một cách nhanh chóng và hiệu quả. Dù không phải lúc nào cũng đảm bảo kết quả tối ưu, giải thuật tham lam được ưa chuộng nhờ sự đơn giản và tốc độ xử lý, đặc biệt trong các bài toán thực tiễn như lập lịch, tìm đường đi ngắn nhất, hay bài toán hành trình của người bán hàng.

2.3.2 Nguyên tắc hoạt động

Giải thuật tham lam hoạt động dựa trên ba nguyên tắc cốt lõi sau đây, tạo nền tảng cho việc tìm ra giải pháp tối ưu trong nhiều bài toán khác nhau:

- **Lựa chọn tối ưu tại từng bước**: Mỗi khi đưa ra quyết định, thuật toán tham lam chỉ xem xét giải pháp tốt nhất có thể tại thời điểm hiện tại, mà không cần phải tính đến toàn bộ không gian giải pháp. Điều này giúp thuật toán đưa ra các quyết định nhanh chóng và hiệu quả mà không bị rối bởi các lựa chọn phức tạp.

- **Không quay lại đánh giá**: Sau khi đưa ra một quyết định, thuật toán tham lam không quay lại để xem xét lại nó, giúp giảm thiểu thời gian tính toán và làm cho quá trình giải quyết bài toán nhanh chóng hơn. Mỗi bước đi được ghi nhớ và không thay đổi, nhờ đó tiết kiệm được tài nguyên tính toán đáng kể.

- **Giải quyết bài toán từng phần**: Thuật toán không cố gắng giải quyết toàn bộ bài toán một lúc, mà chia nhỏ bài toán thành các phần con. Mỗi bước đi tham lam tạo ra một bài toán con, và thuật toán tiếp tục áp dụng nguyên lý tham lam để giải quyết chúng một cách tối ưu.

2.3.3 Điều kiện áp dụng

Để thuật toán tham lam có thể mang lại kết quả tối ưu hoặc gần tối ưu, bài toán cần phải thỏa mãn các điều kiện sau:

- **Tính chất tham lam**: Mỗi quyết định tham lam tại một bước đi phải dẫn đến một phần của giải pháp tối ưu toàn cục. Điều này có nghĩa là mỗi lựa chọn tại mỗi bước phải không làm mất đi cơ hội tìm ra kết quả tối ưu khi kết hợp với các quyết định sau đó.

- **Cấu trúc tối ưu con**: Giải pháp tối ưu của bài toán lớn có thể được xây dựng từ các giải pháp tối ưu của các bài toán con. Chính vì vậy, thuật toán tham lam có thể giải quyết bài toán lớn bằng cách giải quyết các bài toán nhỏ và kết hợp chúng lại để hình thành giải pháp tổng thể.

- **Tính khả thi**: Mỗi lựa chọn tham lam phải đảm bảo khả thi, tức là thỏa mãn các ràng buộc của bài toán. Một quyết định không khả thi sẽ không thể dẫn đến giải pháp hợp lệ, vì vậy mỗi bước đi cần phải kiểm tra tính khả thi trước khi thực hiện.

2.3.4 Ưu điểm và nhược điểm

Ưu điểm của thuật toán tham lam

Thuật toán tham lam (Greedy Algorithm) là một trong những phương pháp giải quyết bài toán tối ưu đơn giản và phổ biến, được ứng dụng rộng rãi nhờ vào các ưu điểm nổi bật sau:

1. **Đơn giản và dễ hiểu**: Thuật toán tham lam có nguyên lý hoạt động rõ ràng và trực quan. Việc xây dựng và triển khai thường rất dễ dàng, giúp tiết kiệm thời gian phát triển, đặc biệt phù hợp với các bài toán quy mô nhỏ hoặc vừa.

2. **Hiệu suất tính toán cao**: Do chỉ tập trung vào việc chọn lựa phương án tối ưu tại mỗi bước (theo hướng tham lam), thuật toán này thường có thời gian thực thi rất nhanh, giúp giảm tải tài nguyên tính toán so với các phương pháp khác như quy hoạch động hay quay lui.

3. **Áp dụng hiệu quả trong một số bài toán nhất định**: Trong những bài toán có tính chất "con đường tối ưu toàn cục bao gồm các bước tối ưu cục bộ", thuật toán tham lam có thể tìm ra lời giải chính xác một cách hiệu quả và nhanh chóng, ví dụ như bài toán cây khung nhỏ nhất, bài toán chọn hoạt động, v.v.

4. **Dễ kết hợp và mở rộng**: Các thuật toán tham lam có thể được tích hợp làm thành phần trong các thuật toán phức tạp hơn hoặc dùng để khởi tạo nghiệm ban đầu cho các phương pháp tối ưu khác.

Những ưu điểm này khiến thuật toán tham lam trở thành một lựa chọn phù hợp trong nhiều tình huống tối ưu hóa cụ thể.

Nhược điểm của thuật toán tham lam

Mặc dù có nhiều ưu điểm, thuật toán tham lam vẫn tồn tại một số hạn chế cần lưu ý khi áp dụng:

1. **Không đảm bảo tìm ra lời giải tối ưu toàn cục**: Do chỉ xét các lựa chọn tối ưu cục bộ tại mỗi bước mà không xem xét toàn cục, thuật toán có thể bỏ qua những phương án tốt hơn về sau và dẫn đến lời giải không tối ưu.

2. **Phụ thuộc mạnh vào cấu trúc bài toán**: Thuật toán chỉ hoạt động hiệu quả nếu bài toán thỏa mãn điều kiện tối ưu con. Nếu không, việc sử dụng phương pháp tham lam có thể cho kết quả sai lệch.

3. **Thiếu khả năng quay lui hay điều chỉnh**: Một khi đã chọn một phương án, thuật toán không quay lại để kiểm tra hay điều chỉnh các bước trước đó. Điều này làm hạn chế khả năng thích ứng với những thay đổi trong quá trình tìm kiếm.

4. **Khó áp dụng cho các bài toán phức tạp hoặc có ràng buộc chặt chẽ**: Trong những bài toán có không gian tìm kiếm rộng hoặc cấu trúc phức tạp, thuật toán tham lam thường không đủ mạnh để tìm ra nghiệm tốt hoặc đáp ứng các điều kiện ràng buộc.

Dù tồn tại những nhược điểm trên, thuật toán tham lam vẫn là một công cụ quan trọng trong hộp công cụ của các nhà khoa học máy tính và nhà tối ưu hóa, đặc biệt khi kết hợp với các phương pháp khác để nâng cao hiệu quả giải quyết bài toán.

2.4 Thuật toán nhánh cận

2.4.1 Giới thiệu

Giải thuật Nhánh và Cận (Branch and Bound) là một phương pháp mạnh mẽ để giải quyết các bài toán tối ưu hóa dạng liệt kê cấu hình. Bằng cách kết hợp chiến lược phân nhánh với việc sử dụng các giới hạn (cận) để loại bỏ sớm các phương án không tiềm năng, thuật toán giúp tìm ra lời giải tối ưu (tối đa hoặc tối thiểu) một cách hiệu quả. Giải thuật này được ứng dụng rộng rãi trong các bài toán như bài toán người du lịch, bài toán ba lô, hoặc tối ưu hóa tổ hợp, nơi không gian lời giải rất lớn và cần một cách tiếp cận thông minh để giảm thiểu việc duyệt toàn bộ khả năng.

2.4.2 Nguyên lý hoạt động

Giải thuật Nhánh Cận hoạt động dựa trên nguyên lý phân nhánh và đánh giá cận như sau:

- **Phân nhánh (Branch)**: Thuật toán chia nhỏ không gian lời giải thành các nhánh, mỗi nhánh đại diện cho một tập hợp các cấu hình con. Không gian lời giải được biểu diễn dưới dạng một tập hợp $D = \{(x_1, x_2, \ldots, x_n)\}$, trong đó mỗi cấu hình $x = (x_1, x_2, \ldots, x_n)$ tương ứng với một giá trị hàm chi phí $f(x)$.
- **Đánh giá cận (Bound)**: Dựa trên hàm chi phí, thuật toán tính toán cận trên hoặc cận dưới của mỗi nhánh. Nếu giá trị của cận cho thấy rằng nhánh đó không thể dẫn đến một lời giải tốt hơn lời giải hiện tại, nhánh đó sẽ bị loại bỏ.

Quy trình thuật toán có thể tóm tắt như sau:

1. Bắt đầu với một cấu hình ban đầu và dần dần xây dựng các thành phần x_i.

2. Tại mỗi bước, tính toán giá trị hàm chi phí $f(x)$ cho cấu hình hiện tại.

3. Nếu cấu hình có tiềm năng trở thành lời giải tối ưu (tốt hơn hoặc bằng lời giải hiện tại), tiếp tục phát triển nhánh bằng cách xác định x_{i+1}.

4. Nếu cấu hình không khả thi hoặc có giá trị hàm chi phí tệ hơn, loại bỏ nhánh đó và quay lui để thử các giá trị khác cho x_{i-1}.

Mục tiêu của giải thuật là tìm ra cấu hình $x = (x_1, x_2, \ldots, x_n)$ sao cho hàm chi phí $f(x)$ đạt giá trị tối ưu, tức là tối đa hoặc tối thiểu hoặc tìm một giá trị nào đó tùy theo bài toán cụ thể.

2.4.3 Điều kiện áp dụng

Để thuật toán Nhánh Cận hoạt động hiệu quả, bài toán cần thỏa mãn các điều kiện sau:

- **Hàm chi phí rõ ràng**: Bài toán phải có một hàm chi phí $f(x)$ được định nghĩa rõ ràng, cho phép tính toán và đánh giá các giá trị tối ưu (tối đa hoặc tối thiểu hoặc giá trị cụ thể).

- **Cấu trúc không gian lời giải**: Không gian lời giải phải có thể biểu diễn dưới dạng các cấu hình liệt kê, tức là tập hợp các phần tử $D = \{(x_1, x_2, \ldots, x_n)\}$, để việc phân nhánh và tìm kiếm trở nên dễ dàng.

- **Cận khả thi**: Phải có cách tính toán cận trên hoặc cận dưới cho mỗi nhánh, nhằm đảm bảo rằng các nhánh không tiềm năng sẽ bị loại bỏ sớm mà không làm mất đi lời giải tối ưu.

- **Tính khả thi của quay lui**: Bài toán cần hỗ trợ cơ chế quay lui, để khi gặp nhánh không khả thi, thuật toán có thể quay lại và thử các giá trị khác cho các cấu hình trước đó.

2.4.4 Ưu điểm và nhược điểm

Ưu điểm của thuật toán nhánh cận

Giải thuật Nhánh Cận (Branch and Bound) là một trong những phương pháp mạnh mẽ để giải quyết các bài toán tối ưu hóa tổ hợp. Với khả năng tìm ra lời giải tối ưu toàn cục, thuật toán này có một số ưu điểm nổi bật như sau:

1. **Tìm lời giải tối ưu toàn cục**: Giải thuật Nhánh và Cận đảm bảo rằng lời giải cuối cùng luôn là tối ưu, không chỉ đối với một phần của bài toán mà còn đối với toàn bộ không gian tìm kiếm. Điều này đặc biệt quan trọng trong các bài toán tối ưu hóa mà độ chính xác là yếu tố quan trọng, như bài toán người du lịch (TSP) hay bài toán ba lô.

2. **Giảm thiểu không gian tìm kiếm**: Nhờ vào việc sử dụng chiến lược phân nhánh và đánh giá cận, thuật toán có thể loại bỏ sớm các nhánh không khả thi hoặc không tối ưu. Điều này giúp giảm đáng kể không gian tìm kiếm và thời gian tính toán so với các phương pháp như tìm kiếm vét cạn.

3. **Linh hoạt và dễ mở rộng**: Giải thuật Nhánh và Cận có thể áp dụng cho nhiều loại bài toán tối ưu hóa tổ hợp khác nhau, bao gồm các bài toán có không gian tìm kiếm phức tạp. Thêm vào đó, thuật toán có thể dễ dàng kết hợp với các phương pháp tối ưu hóa khác để giải quyết các bài toán phức tạp hơn.

4. **Áp dụng hiệu quả trong bài toán NP-Hard**: Giải thuật Nhánh và Cận được sử dụng rộng rãi trong các bài toán NP-Hard, nơi không có các phương pháp giải quyết nhanh và chính xác. Thuật toán này đặc biệt hữu ích trong việc đảm bảo rằng không bỏ sót lời giải tối ưu trong các bài toán tổ hợp.

Những ưu điểm này giúp thuật toán Nhánh và Cận trở thành một công cụ mạnh mẽ và hiệu quả trong việc giải quyết các bài toán tối ưu hóa phức tạp.

Nhược điểm của thuật toán Nhánh Cận

Mặc dù giải thuật Nhánh và Cận có nhiều ưu điểm, tuy nhiên vẫn tồn tại một số nhược điểm cần lưu ý khi áp dụng trong thực tế:

1. **Độ phức tạp tính toán cao**: Giải thuật Nhánh và Cận có thể gặp phải vấn đề về thời gian tính toán, đặc biệt trong các bài toán có không gian tìm kiếm rất lớn. Việc phân nhánh và tính toán các cận cho mỗi nhánh có thể làm tăng đáng kể thời gian thực thi, đặc biệt khi không có chiến lược cận hiệu quả.

2. **Phụ thuộc vào cách thiết kế cận**: Hiệu quả của giải thuật Nhánh và Cận phụ thuộc rất lớn vào cách thiết kế các cận trên và cận dưới cho mỗi nhánh. Nếu các cận không đủ chặt chẽ, thuật toán có thể không giảm được không gian tìm kiếm đáng kể, làm giảm hiệu suất của thuật toán.

3. **Khó khăn trong việc xử lý không gian tìm kiếm lớn**: Trong những bài toán có không gian tìm kiếm rộng lớn và phức tạp, thuật toán Nhánh và Cận có thể gặp phải khó khăn trong việc duy trì hiệu suất tính toán, dẫn đến việc thực thi có thể rất lâu hoặc không hiệu quả.

4. **Cần chiến lược phân nhánh và cận hợp lý**: Để thuật toán hoạt động hiệu quả, việc phân nhánh và tính toán các cận phải được thực hiện một cách hợp lý. Nếu không có chiến lược phân nhánh và cận hợp lý, thuật toán có thể không đạt được hiệu quả tối ưu hoặc thậm chí gặp phải tình trạng không thể giải quyết được bài toán trong thời gian hợp lý.

Chương 3

Triển khai thuật toán

Trong chương này, chúng ta sẽ đi sâu vào việc triển khai bốn thuật toán chính để giải bài toán Người Du Lịch (TSP): Thuật toán Tham lam (Greedy), Thuật toán nhánh cận (Branch and Bound), Thuật toán Tối ưu Bầy Kiến (Ant Colony Optimization - ACO), và Thuật toán Tối ưu Bầy Đàn (Particle Swarm Optimization - PSO). Mỗi thuật toán sẽ được trình bày với các phần phụ bao gồm giới thiệu, mô tả chi tiết, ví dụ minh họa, các lưu ý triển khai, và mã giả để hỗ trợ việc hiểu và áp dụng thực tế.

3.1 Dữ liệu đầu vào

Trong báo cáo này, nhóm sử dụng một tập hợp các tệp dữ liệu có tên `cities_N`, với $N \in \{8, 16, 20, 22, 32, 40, 48, 51, 64\}$. Mỗi tệp tương ứng với một bài toán TSP có N thành phố. Sự lựa chọn này nhằm mục đích kiểm tra hiệu năng và khả năng mở rộng của các thuật toán tối ưu theo quy mô bài toán tăng dần.

Dữ liệu trong mỗi tệp được tổ chức dưới dạng danh sách các cặp tọa độ (x, y), trong đó mỗi cặp biểu diễn vị trí của một thành phố trong mặt phẳng hai chiều. Từ danh sách tọa độ này, ta xây dựng ma trận khoảng cách giữa các thành phố dựa trên công thức Euclid như sau:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Với d_{ij} là khoảng cách giữa thành phố i và thành phố j, (x_i, y_i) và (x_j, y_j) lần lượt là tọa độ của hai thành phố tương ứng.

3.2 Thuật toán ACO

3.2.1 Mô tả chi tiết

1. **Khởi tạo**:
   - Đặt các con kiến vào các vị trí ngẫu nhiên.
   - Khởi tạo pheromone trên các cạnh.
2. **Xây dựng tuyến đường**:
   - Mỗi con kiến xây dựng một tuyến đường bằng cách chọn thành phố tiếp theo dựa trên xác suất phụ thuộc vào pheromone và khoảng cách.
3. **Cập nhật pheromone**:
   - Sau khi tất cả kiến hoàn thành tuyến đường, cập nhật pheromone trên các cạnh dựa trên chất lượng của tuyến đường (độ dài).
4. **Lặp lại**:
   - Lặp lại quá trình xây dựng tuyến đường và cập nhật pheromone cho đến khi đạt số chu kỳ tối đa hoặc điều kiện dừng.

3.2.2 Mã giả

```
# Khởi tạo
Đặt các con kiến vào các vị trí ngẫu nhiên

# Lặp lại cho đến khi đạt số chu kỳ tối đa
REPEAT
    FOR mỗi con kiến
        # Chọn đoạn đường tiếp theo dựa trên xác suất
        Chọn đoạn đường tiếp theo giữa hai thành phố dựa trên pheromone
        và độ dài đoạn đường

        # Kiểm tra nếu tuyến đường hợp lệ
        IF tuyến đường hợp lệ THEN
            # Thêm đoạn đường vào hành trình của con kiến
```

```
            Thêm đoạn đường vào hành trình của con kiến
        END IF

        # Kiểm tra nếu hành trình chưa hoàn thành
        IF hành trình chưa hoàn thành THEN
            Tiếp tục chọn đoạn đường tiếp theo
        END IF
    END FOR

    # Cập nhật pheromone sau mỗi chu kỳ
    Cập nhật pheromone theo độ dài hành trình của mỗi con kiến

UNTIL đạt số chu kỳ tối đa (Nc_max)
```

3.3 Thuật toán PSO

3.3.1 Mô tả chi tiết

1. **Khởi tạo**:
   - Khởi tạo vị trí và vận tốc của các thể (particles) một cách ngẫu nhiên.
2. **Đánh giá**:
   - Tính giá trị thích nghi (fitness) cho mỗi thể dựa trên tuyến đường hiện tại.
3. **Cập nhật vị trí và vận tốc**:
   - Cập nhật vận tốc và vị trí của mỗi thể dựa trên vị trí tốt nhất của nó (Pbest) và vị trí tốt nhất của bầy đàn (Gbest).
4. **Lặp lại**:
   - Lặp lại quá trình đánh giá và cập nhật cho đến khi đạt số lần lặp tối đa hoặc điều kiện dừng.

3.3.2 Mã giả

Thuật toán Tối ưu Bầy Đàn (PSO) cũng đã được mô tả trong chương lý thuyết. Dưới đây là mã giả triển khai PSO cho bài toán TSP:

```
Phương pháp:
Khởi tạo vị trí, vận tốc của các thể;
While chưa tìm được đường đi qua các đỉnh Do
    Begin
        Tính giá trị thích nghi như công thức (2.4);
        Tính Pbest như công thức (2.5);
        Tính Gbest như công thức (2.6);
        Tính vận tốc và cập nhật lại vị trí như công thức (2.7) và (2.8);
    End
Hiển thị ra đường đi gần tối ưu qua các đỉnh
```

3.4 Thuật toán Tham lam (Greedy)

Trong phần này, chúng tôi trình bày thuật toán tham lam (Greedy Algorithm) áp dụng cho bài toán TSP. Thuật toán này sử dụng chiến lược đơn giản: tại mỗi bước, người du lịch chọn thành phố chưa thăm gần nhất từ vị trí hiện tại, cho đến khi tất cả các thành phố được thăm và quay về điểm xuất phát.

3.4.1 Mô tả chi tiết

1. **Khởi tạo**: Chọn một thành phố khởi đầu ngẫu nhiên và đánh dấu là đã thăm.
2. **Lặp lại**:
    - Từ thành phố hiện tại, tìm thành phố chưa thăm có khoảng cách ngắn nhất.
    - Thêm thành phố này vào tuyến đường và đánh dấu là đã thăm.
    - Cập nhật thành phố hiện tại thành thành phố vừa thêm.
3. **Hoàn thành chu trình**: Sau khi tất cả thành phố đã được thăm, thêm thành phố khởi đầu vào cuối tuyến đường để hoàn thành chu trình Hamilton.

3.4.2 Lưu ý triển khai

- Cần một cấu trúc dữ liệu để theo dõi các các thành phố đã thăm (ví dụ: mảng boolean).
- Cần một cách để tìm thành phố chưa thăm gần nhất từ thành phố hiện tại một cách hiệu quả.

3.4.3 Mã giả

```
Khởi tạo:
    Chọn một thành phố khởi đầu ngẫu nhiên (ví dụ: thành phố 1).
    Đánh dấu thành phố này là đã thăm.
    Thêm thành phố này vào tuyến đường.

WHILE còn thành phố chưa thăm:
    Từ thành phố hiện tại, tìm thành phố chưa thăm gần nhất dựa trên
    khoảng cách.
    Thêm thành phố này vào tuyến đường.
    Đánh dấu thành phố này là đã thăm.
    Cập nhật thành phố hiện tại thành thành phố vừa thêm.

Thêm thành phố khởi đầu vào cuối tuyến đường để hoàn thành chu trình Hamilton.
Trả về tuyến đường và tổng chi phí (cost).
```

3.5 Thuật toán Nhánh Cận (Branch and Bound)

Thuật toán Nhánh Cận là một phương pháp tìm kiếm có hệ thống để tìm ra giải pháp tối ưu cho bài toán TSP bằng cách phân chia không gian tìm kiếm và loại bỏ các nhánh không khả thi.

3.5.1 Mô tả chi tiết

1. **Khởi tạo**:

- Bắt đầu với một tập hợp các tuyến đường khả thi, ban đầu là tuyến đường chỉ chứa thành phố khởi đầu.

2. **Phân nhánh**:
   - Mở rộng tuyến đường bằng cách thêm một thành phố chưa thăm vào tuyến đường hiện tại.
   - Tạo ra các nhánh mới cho mỗi lựa chọn thành phố tiếp theo.
3. **Đánh giá cận**:
   - Tính toán cận dưới cho mỗi nhánh để ước lượng chi phí tối thiểu có thể đạt được.
   - Nếu cận dưới của một nhánh lớn hơn chi phí của giải pháp tốt nhất hiện tại, loại bỏ nhánh đó.
4. **Tìm kiếm**:
   - Tiếp tục mở rộng các nhánh có triển vọng nhất cho đến khi tìm ra tuyến đường hoàn chỉnh với chi phí nhỏ nhất.

3.5.2 Lưu ý triển khai

- Cần một cách để tính toán cận dưới hiệu quả, ví dụ như sử dụng ma trận khoảng cách giảm.
- Cần một hàng đợi ưu tiên để quản lý các nhánh theo thứ tự triển vọng.

3.5.3 Mã giả

```
Khởi tạo:
    Tạo một hàng đợi ưu tiên với tuyến đường ban đầu chỉ chứa thành phố khởi đầu.
    Đặt giải pháp tốt nhất là vô cùng lớn.

WHILE hàng đợi không rỗng:
    Lấy nhánh có cận dưới nhỏ nhất từ hàng đợi.
    IF nhánh này là tuyến đường hoàn chỉnh THEN
        IF chi phí của tuyến đường < giải pháp tốt nhất THEN
```

```
            Cập nhật giải pháp tốt nhất.
        END IF
    ELSE
        FOR mỗi thành phố chưa thăm:
            Tạo nhánh mới bằng cách thêm thành phố này vào tuyến đường.
            Tính toán cận dưới cho nhánh mới.
            IF cận dưới < giải pháp tốt nhất THEN
                Thêm nhánh mới vào hàng đợi.
            END IF
        END FOR
    END IF

Trả về giải pháp tốt nhất.
```

Chương 4

Kết quả thực nghiệm:

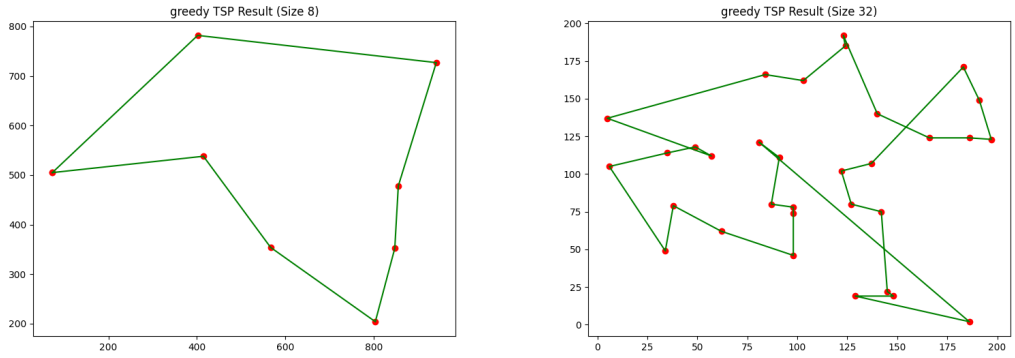
Trong phần này, chúng tôi trình bày các kết quả thực nghiệm của bốn thuật toán trên các bộ dữ liệu TSP khác nhau, với số lượng thành phố từ nhỏ đến lớn để đánh giá hiệu quả của từng thuật toán.

4.1 Kết quả thực nghiệm

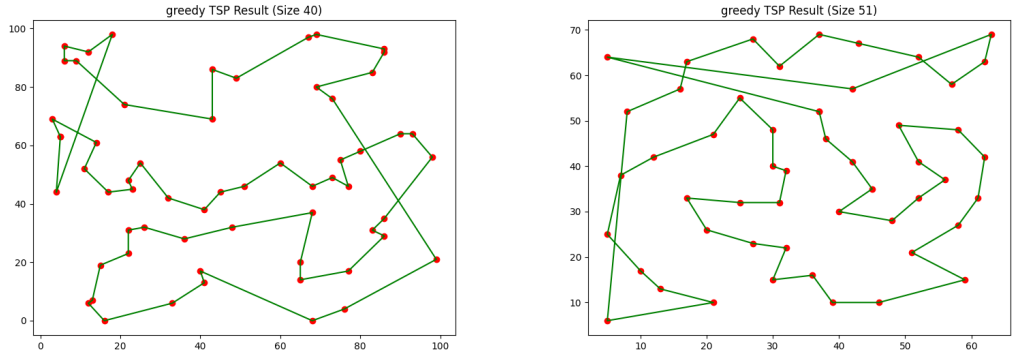
Để đánh giá hiệu quả của các thuật toán triển khai, nhóm tiến hành thực nghiệm trên các tập dữ liệu `cities_N` với $N \in \{8, 16, 20, 22, 32, 40, 48, 51, 64\}$. Các kết quả được thể hiện dưới dạng hình ảnh trực quan bao gồm: lộ trình thu được (route), chi phí (cost), và sự hội tụ theo số lần lặp (iteration) nếu có. Mỗi hình ảnh minh họa quá trình chạy của một thuật toán trên một tập dữ liệu cụ thể.

4.1.1 Thuật toán Greedy

Thuật toán Greedy cho kết quả nhanh nhưng có xu hướng bị kẹt tại nghiệm cục bộ. Dưới đây là một số ví dụ về lộ trình và chi phí trên từng tập dữ liệu:



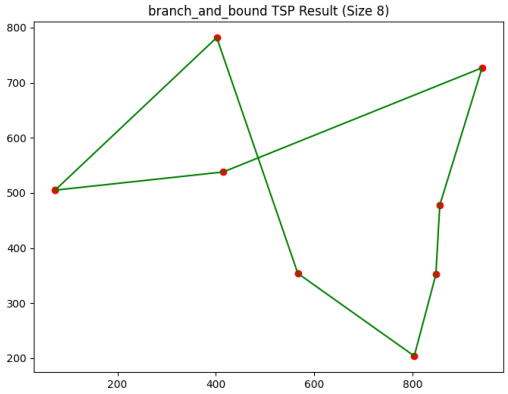
Hình 4.1: Lộ trình Greedy với 8 và 32 thành phố



Hình 4.2: Lộ trình Greedy với 40 và 51 thành phố

4.1.2 Thuật toán Branch and Bound

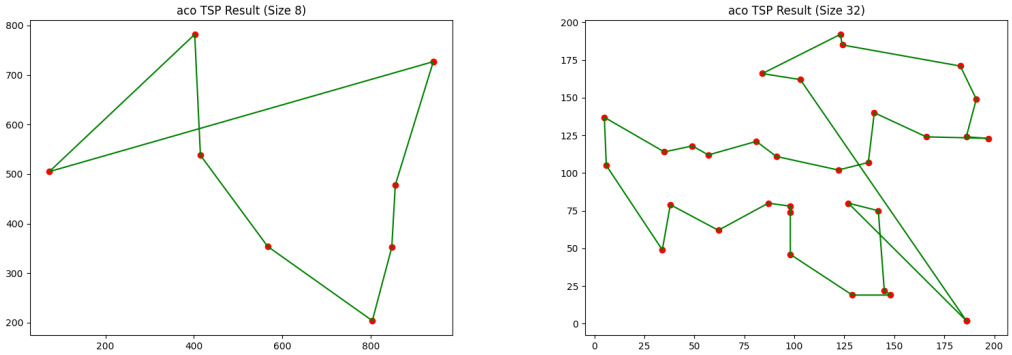
Do độ phức tạp tính toán lớn, thuật toán Branch and Bound chỉ được áp dụng cho tập dữ liệu nhỏ (`cities_8`). Hình dưới đây mô tả lộ trình tối ưu thu được:



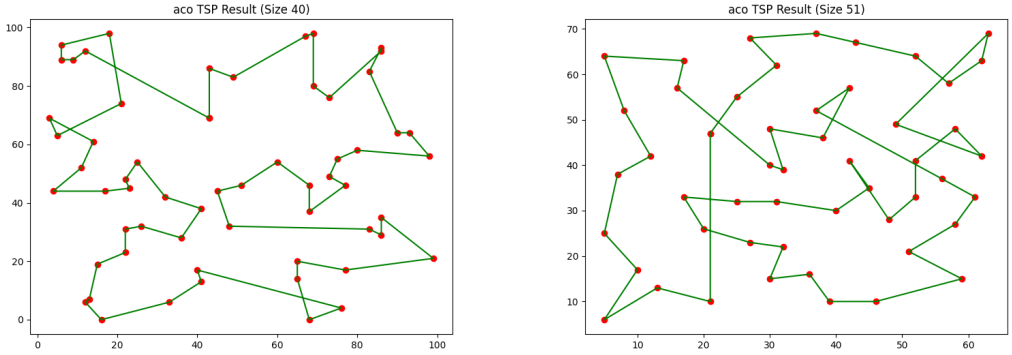
Hình 4.3: Lộ trình tối ưu bằng thuật toán Branch and Bound (8 thành phố)

4.1.3 Thuật toán ACO

Thuật toán ACO thể hiện hiệu quả khá tốt với khả năng cải thiện lời giải qua từng vòng lặp. Dưới đây là kết quả hội tụ và lộ trình cuối cùng trên một số tập dữ liệu:



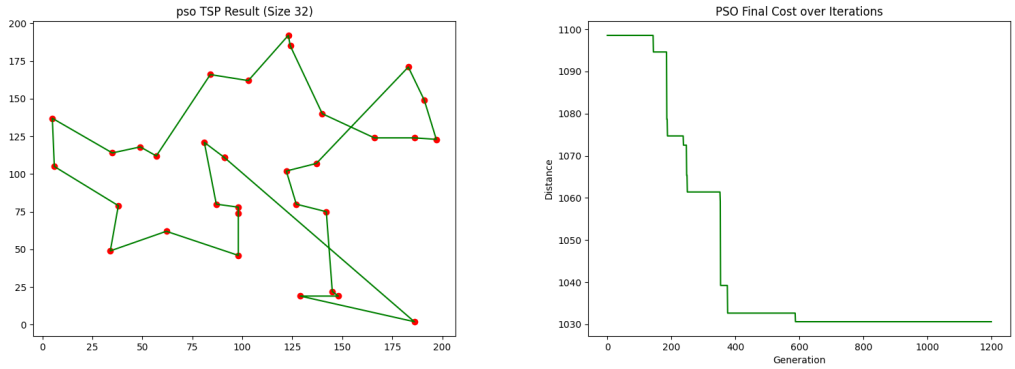
Hình 4.4: Kết quả thuật toán ACO với 8 và 32 thành phố



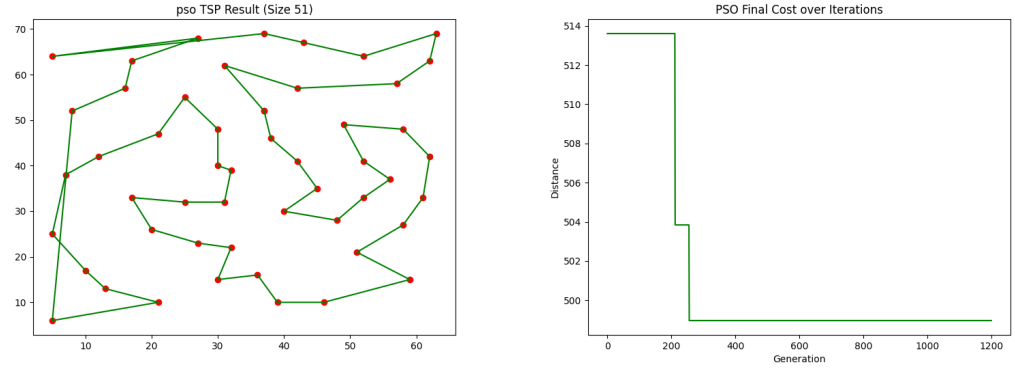
Hình 4.5: Kết quả thuật toán ACO với 40 và 51 thành phố

4.1.4 Thuật toán PSO

Tương tự ACO, thuật toán PSO có khả năng tìm lời giải tốt nhờ cơ chế cập nhật tập thể. Kết quả dưới đây minh họa sự hội tụ và lời giải cuối cùng:



Hình 4.6: Kết quả thuật toán PSO với 32 thành phố: lộ trình và đồ thị hội tụ

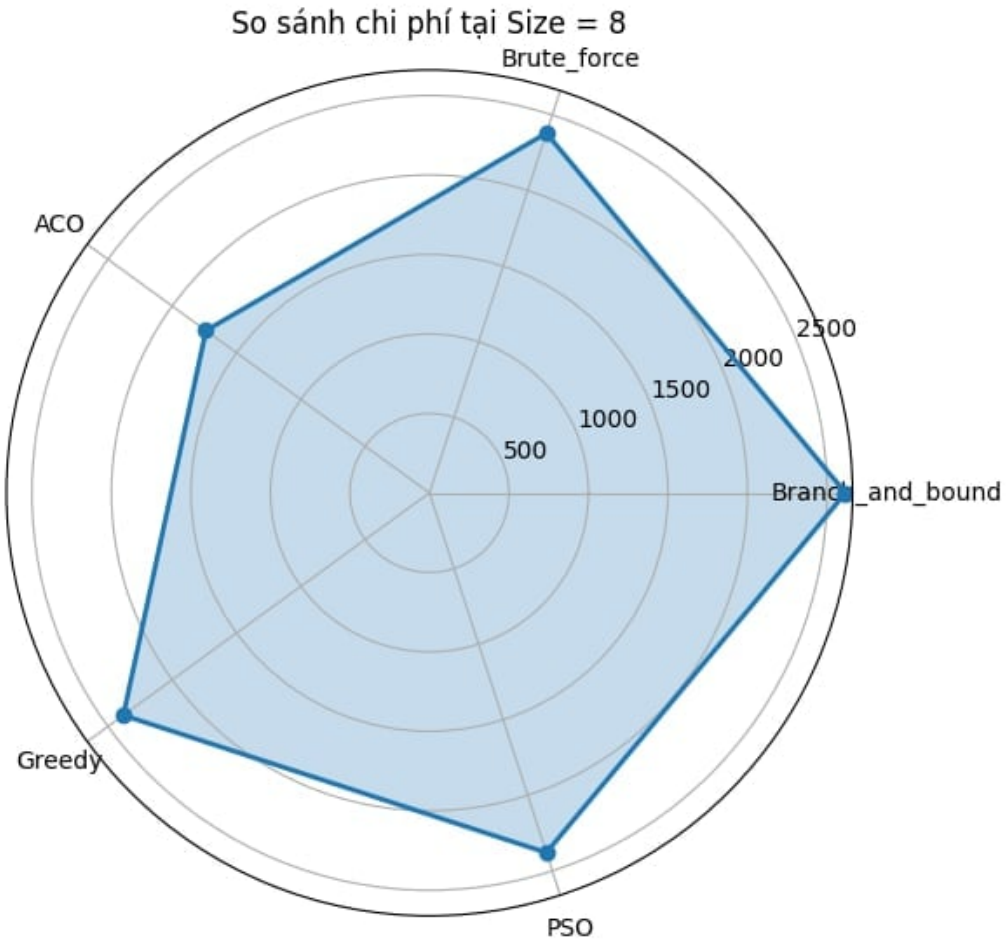


Hình 4.7: Kết quả thuật toán PSO với 51 thành phố: lộ trình và đồ thị hội tụ

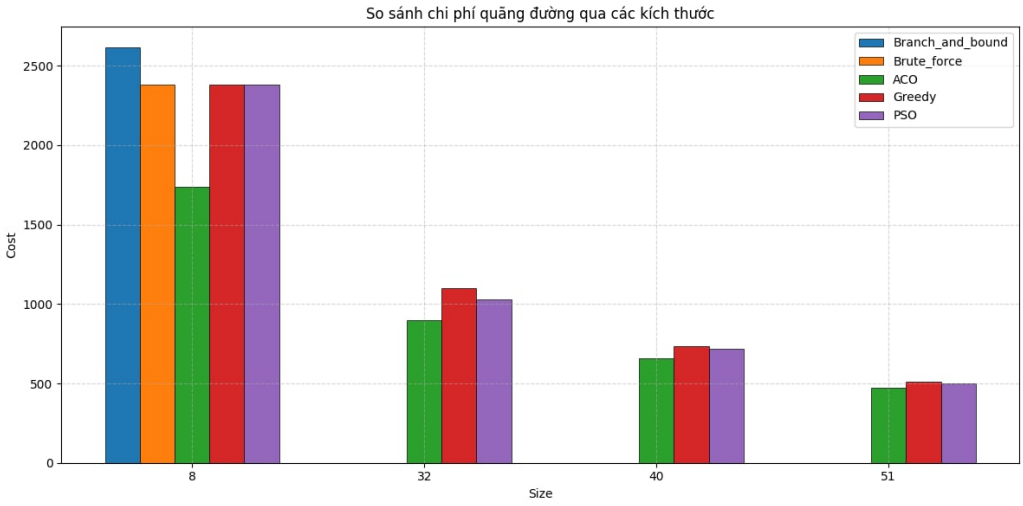
4.2 So sánh các thuật toán

4.2.1 Chi phí

Biểu đồ Hình 4.9 thể hiện sự so sánh chi phí (cost) của các thuật toán giải bài toán TSP trên các bộ dữ liệu có số thành phố khác nhau: 8, 32, 40 và 51. Các thuật toán được so sánh bao gồm: Branch and Bound, Brute Force, ACO, Greedy và PSO.



Hình 4.8: So sánh chi phí quãng đường với kích thước bài toán là



Hình 4.9: So sánh chi phí quãng đường qua các kích thước bài toán

Nhận xét:

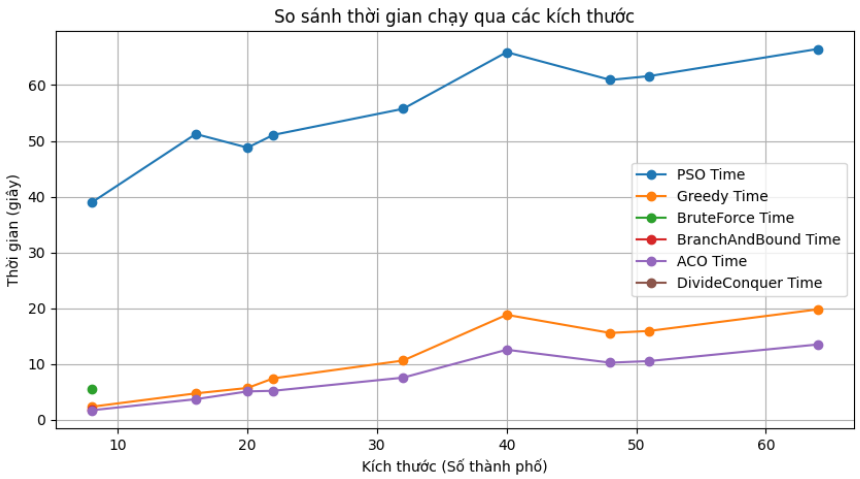
- **Với dữ liệu 8 thành phố:** Các thuật toán nhánh cận, brute force và greedy đều đạt chi phí tương đối cao (trên 2100), trong khi ACO và đặc biệt là PSO cho kết quả tốt hơn rõ rệt với chi phí thấp hơn khoảng 15–20%.

- **Với 32 thành phố:** Greedy cho chi phí cao nhất. ACO và PSO tiếp tục thể hiện hiệu quả vượt trội với chi phí thấp nhất, chênh lệch không đáng kể giữa hai thuật toán này.
- **Với 40 thành phố:** Khoảng cách giữa các thuật toán bắt đầu thu hẹp hơn. Greedy vẫn cho kết quả kém nhất, trong khi ACO và PSO có chi phí gần như tương đương và thấp hơn đáng kể so với Greedy.
- **Với 51 thành phố:** Tất cả các thuật toán đều hội tụ về một mức chi phí gần nhau. Tuy nhiên, ACO và PSO vẫn duy trì ưu thế nhẹ về mặt chi phí so với Greedy.

Tổng kết: Trong tất cả các kích thước bài toán, PSO và ACO luôn nằm trong nhóm có chi phí thấp nhất. Greedy tuy nhanh nhưng thường cho kết quả kém hơn. Các thuật toán chính xác như nhánh cận và brute force chỉ áp dụng được cho quy mô rất nhỏ. PSO tỏ ra hiệu quả nhất trong việc giữ mức chi phí thấp và ổn định khi quy mô bài toán tăng.

4.2.2 Thời gian thực hiện

Biểu đồ Hình 4.10 thể hiện thời gian chạy (tính bằng giây) của các thuật toán giải bài toán TSP theo số lượng thành phố từ 8 đến 64. Các thuật toán được so sánh gồm: Greedy, Brute Force, Branch and Bound, ACO, PSO và Divide and Conquer (nếu có).



Hình 4.10: So sánh thời gian chạy theo kích thước bài toán

Nhận xét:

- **PSO** có thời gian chạy cao hơn các thuật toán khác, dao động từ 39 đến gần 70 giây. Tuy nhiên, cần lưu ý rằng đây là thuật toán được thực thi đầu tiên trong quá

trình thử nghiệm, và thời gian chạy bao gồm cả chi phí khởi tạo ban đầu (đặc biệt là tính toán ma trận khoảng cách giữa các thành phố). Nếu loại trừ bước này, thời gian thực thi thực tế của PSO là hợp lý và tương đương với ACO.

- **Greedy** có thời gian chạy thấp, tăng nhẹ theo số lượng thành phố, từ khoảng 3 đến 20 giây. Đây là thuật toán đơn giản nhưng hiệu quả về mặt thời gian.
- **ACO** có thời gian xử lý tuyến tính với kích thước bài toán, từ 2 đến 13 giây, cho thấy hiệu năng ổn định và phù hợp cho các bài toán vừa và lớn.
- **Branch and Bound** và **Brute Force** chỉ áp dụng được với dữ liệu nhỏ do thời gian tăng nhanh. Với nhiều thành phố, thời gian chạy vượt giới hạn chấp nhận được.

Kết luận tạm thời: - Trong các bài toán TSP quy mô lớn, PSO và ACO có thể chấp nhận được về thời gian nếu cần lời giải gần tối ưu.

- Greedy có ưu điểm về tốc độ, nhưng đánh đổi bằng chất lượng lời giải.

- Các thuật toán như Brute Force, Branch and Bound chỉ thích hợp cho dữ liệu nhỏ do thời gian tăng rất nhanh.

4.3 Tóm tắt và đề xuất cải tiến

Từ kết quả thực nghiệm, có thể thấy rằng:

- Với các bộ dữ liệu nhỏ, thuật toán Branch & Bound là lựa chọn tốt nhất vì nó đảm bảo tìm ra giải pháp tối ưu trong thời gian hợp lý.
- Với các bộ dữ liệu lớn hơn, ACO và PSO là các lựa chọn khả thi, trong đó ACO thường cho kết quả tốt hơn nhưng tốn nhiều thời gian hơn.
- Thuật toán Greedy có thể được sử dụng như một phương pháp nhanh để có giải pháp ban đầu.

Đề xuất cải tiến

Dựa trên kết quả thực nghiệm, nhóm đề xuất một số hướng cải tiến trong tương lai như sau:

- **Tách riêng bước tiền xử lý dữ liệu** (ví dụ: tính ma trận khoảng cách) khỏi phần đo thời gian thuật toán để đảm bảo tính công bằng khi so sánh hiệu năng.

- **Tối ưu hóa thuật toán PSO**: Khai thác các biến thể như Adaptive PSO, Hybrid PSO-ACO để cải thiện tốc độ hội tụ và tránh bẫy cực trị cục bộ.
- **Tinh chỉnh tham số ACO và PSO** tự động bằng phương pháp lưới (grid search) hoặc tiến hóa (genetic tuning) nhằm tối ưu hóa hiệu suất thuật toán theo từng bài toán cụ thể.
- **Áp dụng song song hóa hoặc GPU** (ví dụ: với PSO hoặc ACO) để rút ngắn thời gian thực thi khi mở rộng lên hàng trăm hoặc hàng nghìn thành phố.
- **Mở rộng đánh giá** sang các bộ dữ liệu thực tế (TSPLIB, dữ liệu bản đồ thật) nhằm kiểm tra khả năng áp dụng vào các bài toán logistics, định tuyến thực tế.

Tài liệu tham khảo

[1] Yong Wang and Zunpu Han (August 2021) Ant colony optimization for traveling salesman problem based on parameters optimization
Ant colony optimization for traveling salesman problem based on parameters optimization

[2] Analytics Vidhya. (2021, October). An Introduction to Particle Swarm Optimization Algorithm
An Introduction to Particle Swarm Optimization Algorithm

[3] A. T. B. University (2016) Implementation of Greedy Algorithm in Travel Salesman Problem
Implementation of Greedy Algorithm in Travel Salesman Problem - ResearchGate

[4] S. M. S. R. (2018) Travelling Salesman Problem Using Branch And Bound Technique
Travelling Salesman Problem Using Branch And Bound Technique - IJMTT