BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC XÂY DỰNG HÀ NỘI

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN TỐT NGHIỆP

Đề Tài: Thuật toán mô phỏng tự nhiên giải bài toán tối ưu tổ hợp

Giảng viên hướng dẫn: Phạm Hồng Phong

Sinh viên thực hiện: Nguyễn Trung Hiệp

MSV: 69565

Lớp: 65CS3

Hà Nội - 04/2024

Mục Lục

LỜI NÓI ĐẦU

Như chúng ta đã biết, trong khoảng 10 năm trở lại đây, công nghệ thông tin đã bùng nổ và phát triển mạnh mẽ ở nước ta. Có thể nói sự phát triển như vũ bão của khoa học và công nghệ trong thời gian qua đã tạo ra những sản phẩm công nghệ mới và đem lại rất nhiều lợi ích cho cuộc sống. Nó đang chiếm phần lớn trong việc phục vụ của nhiều ngành nghề cũng như phục vụ đời sống của con người. Cùng với đó là loài người chúng ta luôn muốn tìm hiểu, cải tiến và khắc phục các nhược điểm của các bài toán có trong cuộc sống. Ví dụ như bài toán lập lịch, người bán hàng,… cho nên trong bài báo cáo này em tập trung nghiên cứu về 2 thuật toán được áp dụng nhiều để giải quyết các bài toán tối ưu, cùng với đó là so sánh với các thuật toán khác để thấy được sự khác biệt mà 2 thuật toán mang lại.

Trân trọng.

Phần 1: Tổng quan về đề tài

1.1 Giới thiệu

- Như đã nói ở trên thì bài báo cáo này của em sẽ nghiên cứu về 2 thuật toán chủ đạo đó là thuật toán di truyền (Genetic Algorithms - GA), thuật toán tối ưu hóa đàn kiến (Ant Colony Optimization - ACO).
  a. GA: là một thuật toán tối ưu hóa nguồn cảm hứng từ tự nhiên, mô phỏng quá trình tiến hóa sinh học. Thuật toán này sử dụng các cơ chế di truyền như chọn lọc tự nhiên, lai ghép (crossover), và đột biến (mutation) để tạo ra các thế hệ giải pháp mới từ thế hệ hiện tại. GA bắt đầu từ một quần thể các cá thể (các giải pháp) ngẫu nhiên, sau đó lặp đi lặp lại quá trình chọn lọc, lai ghép, và đột biến để cải thiện chất lượng của quần thể. Thuật toán này phù hợp cho các bài toán tối ưu hóa liên tục hoặc tổ hợp, ví dụ như tối ưu hóa thiết kế, tối ưu hóa đa mục tiêu, và vấn đề phân công
  b. ACO: là một thuật toán tối ưu hóa nguồn cảm hứng từ tự nhiên, dựa trên hành vi tìm kiếm thức ăn của đàn kiến. Thuật toán này được Marco Dorigo đề xuất vào đầu những năm 1990. Trong ACO, các kiến khám phá không gian tìm kiếm và xây dựng các giải pháp dựa trên phép đánh giá chất lượng của các giải pháp và một chất pheromone mà chúng tiết ra. Chất pheromone này giúp hướng dẫn các kiến khác đến các giải pháp tốt hơn. Thuật toán này thường được sử dụng để giải quyết các bài toán tối ưu hóa tổ hợp, chẳng hạn như bài toán người bán hàng (Travelling Salesman Problem - TSP), bài toán lập lịch, và các bài toán tuyến đường.

1.2 Lý do chọn đề tài

- Ứng dụng Rộng Rãi: Cả ACO và GA đều có khả năng ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau như tối ưu hóa trong lĩnh vực logistics, lập lịch sản xuất, vận tải, thiết kế mạng, và nhiều bài toán khoa học khác. Việc tìm hiểu sâu về các thuật toán này có thể mở ra nhiều cơ hội áp dụng thực tế.
- Cải Thiện và Đổi Mới: Cả hai thuật toán đều cho phép không gian lớn để đổi mới và cải tiến. Có thể tập trung vào cải thiện hiệu quả tính toán của chúng, hoặc phát triển các biến thể mới phù hợp hơn với các loại bài toán cụ thể.

- Giải quyết Bài toán Phức Tạp: Thuật toán ACO và GA đặc biệt hữu ích trong việc giải quyết các bài toán tối ưu hóa phức tạp, nơi các phương pháp truyền thống không hiệu quả. Chọn các thuật toán này làm đề tài nghiên cứu có thể giúp giải quyết những thách thức thực tế trong các ngành công nghiệp và nghiên cứu.
- Nhu cầu Thực Tiễn và Thương Mại: Cả hai thuật toán có nhiều ứng dụng thực tế có thể chuyển giao sang các sản phẩm và dịch vụ thương mại. Tìm hiểu về chúng có thể cung cấp nền tảng để phát triển các giải pháp công nghệ cao, góp phần vào sự phát triển kinh tế.
- Tính Đa Dạng và Linh Hoạt: ACO và GA cung cấp một khung tư duy linh hoạt cho các nhà khoa học và kỹ sư để giải quyết nhiều loại bài toán khác nhau. Nghiên cứu về chúng giúp phát triển kỹ năng giải quyết vấn đề và sáng tạo trong nhiều hoàn cảnh khác nhau.
- Cộng Đồng Nghiên Cứu Sôi Nổi: Cả hai lĩnh vực này có một cộng đồng nghiên cứu lớn và sôi động, với nhiều hội nghị, workshop, và tạp chí chuyên ngành, cung cấp cơ hội tốt để trao đổi kiến thức và kết nối với các nhà nghiên cứu khác.

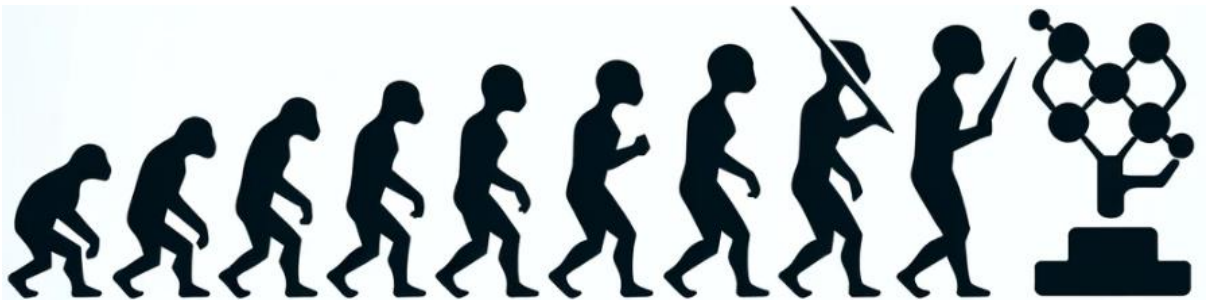
1.3 Phạm vi nghiên cứu

1. Đối Tượng Nghiên Cứu
    - ACO: Tập trung vào giải quyết các bài toán tối ưu hóa tổ hợp, như bài toán người bán hàng (TSP) và bài toán lập lịch.
    - GA: Áp dụng cho các bài toán tối ưu hóa liên tục và đa mục tiêu, chẳng hạn như trong thiết kế kỹ thuật và sinh học tính toán.
2. Cải Tiến Kỹ Thuật
    - Phát triển và thử nghiệm các cải tiến thuật toán nhằm cải thiện hiệu suất, độ chính xác, và thời gian tính toán.
3. Ứng Dụng Thực Tế
    - Phát triển các ứng dụng cụ thể của ACO và GA trong lĩnh vực như robotics và bioinformatics.
4. Phương Pháp Đánh Giá
    - So sánh hiệu quả của ACO và GA với các thuật toán khác trong điều kiện thực nghiệm nhằm đánh giá ưu nhược điểm.
5. Môi Trường Thực Nghiệm
    - Sử dụng các công cụ và môi trường phát triển phổ biến Python để triển khai và kiểm thử thuật toán.

Phần 2: Thuật toán di truyền (GA) và thuật toán tối ưu hóa đàn kiến (ACO)

2.1 Thuật toán di truyền (GA)

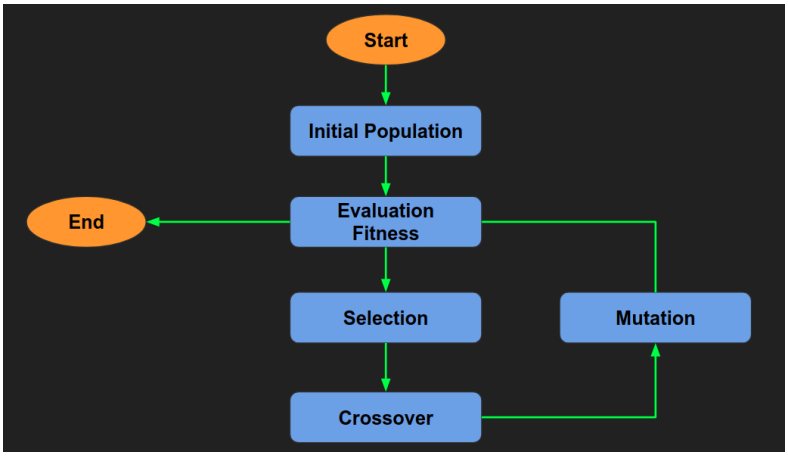
- John Holland, người phát minh ra Thuật toán Di truyền vào những năm 1960 tại Đại học Michigan. Mục tiêu ban đầu của Holland là tạo ra một mô hình máy tính mô phỏng các nguyên tắc của tiến hóa sinh học để nghiên cứu các quá trình thích nghi.
- Nguyên lý cơ bản của Thuật toán Di truyền: Giải thích cách GA mô phỏng quá trình tiến hóa tự nhiên thông qua chọn lọc tự nhiên, lai ghép, và đột biến. Mô tả cách mà GA tạo ra các thế hệ cá thể mới để tìm kiếm các giải pháp tối ưu cho một bài toán cụ thể. Vai trò của di truyền trong việc chuyển giao các đặc tính giữa các thế hệ và vai trò của đột biến trong việc duy trì sự đa dạng di truyền, từ đó giúp thuật toán khám phá ra những iải pháp mới và hiệu quả hơn.



2.1.1 Các thành phần cơ bản của GA

1. Mã hóa (Encoding)
   a. Mục đích: Giải thích tầm quan trọng của việc mã hóa đối với hiệu quả của GA. Mã hóa là quá trình biểu diễn các biến quyết định của bài toán thành một chuỗi, thường được gọi là chromosome.
   b. Các loại mã hóa: Phổ biến nhất là mã hóa nhị phân, nhưng còn có mã hóa số nguyên, số thực, và mã hóa dựa trên đối tượng cho các bài toán phức tạp hơn. Mô tả ưu và nhược điểm của mỗi phương pháp.

2. Khởi tạo quần thể (Population Initialization)
    a. Mô tả quá trình: Quần thể ban đầu của GA thường được sinh ra một cách ngẫu nhiên để đảm bảo sự đa dạng. Quần thể gồm nhiều cá thể, mỗi cá thể đại diện cho một giải pháp tiềm năng.
    b. Tầm quan trọng: Khởi tạo quần thể là bước đầu tiên và quan trọng để khám phá không gian giải pháp. Số lượng cá thể trong quần thể có ảnh hưởng đến khả năng tìm kiếm và hiệu suất của GA.
3. Hàm thích nghi (Fitness Function)
    a. Xác định: Hàm thích nghi là hàm đánh giá mức độ "thích hợp" của mỗi cá thể trong quần thể, thường dựa trên chất lượng của giải pháp mà cá thể đó biểu diễn.
    b. Vai trò: Hàm thích nghi quyết định cá thể nào sẽ "sống sót" và được chọn để sinh sản trong thế hệ tiếp theo. Hàm thích nghi phải được thiết kế cẩn thận để đảm bảo rằng nó phản ánh chính xác mục tiêu của bài toán.
4. Chọn lọc (Selection)
    a. Mục đích: Chọn lọc là quá trình chọn ra cá thể từ quần thể hiện tại để tạo ra cá thể cho quần thể thế hệ tiếp theo.
    b. Phương pháp: Giới thiệu các kỹ thuật chọn lọc phổ biến như Roulette Wheel, Tournament Selection, và Stochastic Universal Sampling. Mỗi phương pháp có ưu và nhược điểm riêng trong cách thức tạo đa dạng di truyền.
5. Lai ghép (Crossover)
    a. Xác định: Lai ghép là quá trình kết hợp đặc điểm của hai cá thể cha mẹ để tạo ra cá thể con.
    b. Cách thực hiện: Mô tả các kiểu lai ghép như One-point Crossover, Two-point Crossover, và Uniform Crossover, cùng với ảnh hưởng của chúng đến đặc điểm di truyền của quần thể.
6. Đột biến (Mutation)
    a. Vai trò: Đột biến giúp duy trì sự đa dạng di truyền trong quần thể bằng cách thay đổi ngẫu nhiên gen của cá thể.
    b. Phương pháp: Giới thiệu cách thức và tỷ lệ đột biến thường được sử dụng, bao gồm Bit-flip Mutation cho mã hóa nhị phân và các phương pháp khác cho số nguyên hoặc số thực.



```
START
Generate the initial population
Compute fitness
REPEAT
    Selection
    Crossover
    Mutation
    Compute fitness
UNTIL population has converged
STOP
```

Hình 6 Mã giả

2.1.2 Các tính chất của GA

1) **Tính chính xác (Accuracy):** GA là thuật toán tìm kiếm xác suất, nghĩa là nó không đảm bảo luôn tìm thấy giải pháp tối ưu toàn cục nhưng lại có khả năng tìm ra giải pháp tối ưu cục bộ hoặc gần với tối ưu toàn cục. Độ chính xác của GA phụ thuộc vào các yếu tố như hàm thích nghi, thiết kế nhiễm sắc thể, và cách thức áp dụng các toán tử di truyền như lai ghép và đột biến.
2) **Tính khách quan (Objectivity):** GA hoạt động dựa trên nguyên tắc chọn lọc tự nhiên, làm cho kết quả của thuật toán không bị ảnh hưởng bởi sự thiên vị cá nhân hoặc yếu tố ngoại cảnh. Mọi cá thể trong quần thể đều có cơ hội được đánh giá một cách khách quan qua hàm thích nghi của chúng, đảm bảo tính công bằng trong quá trình tuyển chọn.
3) **Tính phổ dụng (Generality):** GA là một phương pháp rất linh hoạt và có thể được áp dụng cho nhiều loại bài toán tối ưu hóa khác nhau, từ các bài toán liên quan đến tối ưu hóa số, tối ưu hóa kết cấu, đến các bài toán phân lập và lập lịch. Sự phổ dụng này làm cho GA trở thành một công cụ ưa thích trong nhiều ngành nghề khác nhau.
4) **Tính rõ ràng (Clarity):** Quy trình của GA rất rõ ràng và có hệ thống, bao gồm khởi tạo quần thể, đánh giá hàm thích nghi, lai ghép, đột biến, và chọn lọc. Mỗi bước trong thuật toán có mục đích rõ ràng và được thực hiện theo một quy trình xác định, giúp người sử dụng dễ dàng theo dõi và hiểu được quá trình tối ưu hóa đang diễn ra.
5) **Tính kết thúc (Termination):** GA thường sử dụng một hoặc nhiều tiêu chí để kết thúc quá trình tìm kiếm, bao gồm đạt được số lượng thế hệ tối đa, không có sự cải thiện đáng kể trong hàm thích nghi qua các thế hệ, hoặc đạt được một ngưỡng giá trị hàm thích nghi nhất định. Tiêu chí kết thúc giúp kiểm soát quá trình tối ưu và đảm bảo rằng thuật toán dừng lại khi đã đạt được kết quả mong muốn hoặc khi đã đủ tốt.

2.1.3 Độ phức tạp

Độ phức tạp của thuật toán Di truyền (Genetic Algorithm - GA) thường không thể xác định một cách chính xác và duy nhất vì nó phụ thuộc vào nhiều yếu tố khác nhau, bao gồm thiết kế của nhiễm sắc thể, hàm thích nghi, kích thước quần thể, số lượng thế hệ, và các toán tử di truyền như lai ghép và đột biến.

Độ phức tạp về thời gian

Độ phức tạp về thời gian của GA thường được xem xét qua ba yếu tố chính: kích thước quần thể (N), số lượng thế hệ (G), và độ phức tạp của hàm đánh giá (f). Độ phức tạp tổng thể thường được biểu diễn qua công thức:

$$O(N \times G \times f)$$

Ở đây:

- N là số lượng cá thể trong quần thể.
- G là số thế hệ mà thuật toán chạy.
- f là độ phức tạp của hàm thích nghi có thể là O(1), O(n), $O(n^2)$, ...

Độ phức tạp về không gian

Độ phức tạp về không gian của GA phụ thuộc chủ yếu vào kích thước quần thể và biểu diễn của nhiễm sắc thể. Mỗi cá thể trong quần thể có thể yêu cầu một lượng nhất định bộ nhớ để lưu trữ thông tin gen của nó, và do đó độ phức tạp về không gian cũng có thể được biểu thị qua kích thước quần thể:

$$O(N \times s)$$

Ở đây:

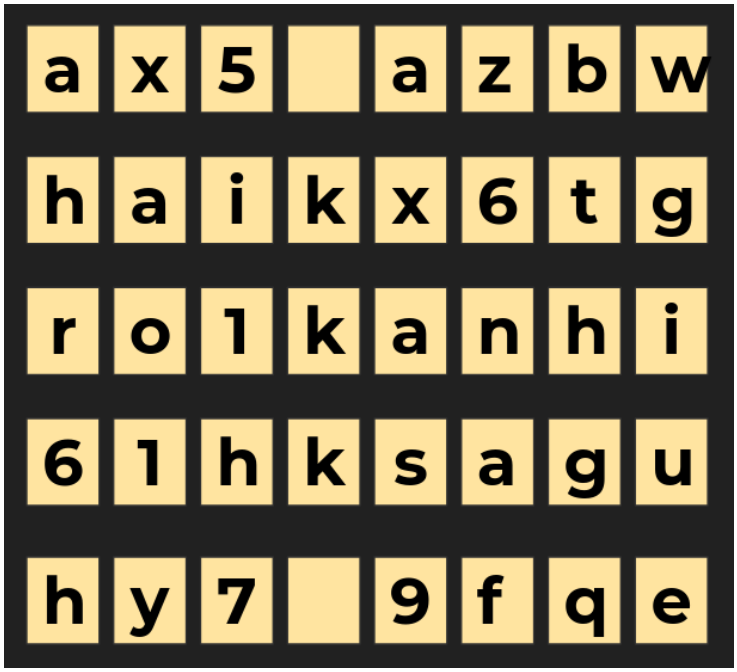
- s là không gian lưu trữ cần thiết cho mỗi cá thể, kể cả các cá thể con được sinh ra.

2.1.4 Bài toán điển hình

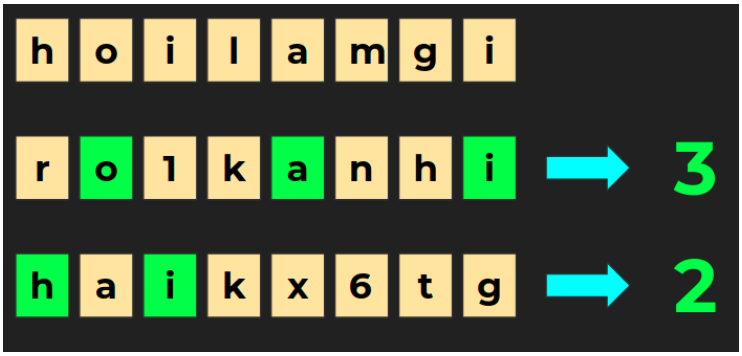
Bài toán Đoán Mật Khẩu (Password Guessing Problem): Trong bài toán này, mục tiêu là tìm ra một chuỗi ký tự (mật khẩu) bằng cách 'đoán' liên tục. GA sử dụng các toán tử di truyền như lai ghép và đột biến để tạo ra các thế hệ mới của các chuỗi ký tự, tiến tới mật khẩu đúng. Đây là một minh họa rất trực quan và hiệu quả của cách mà GA tối ưu hóa và tiến hóa dần tới giải pháp qua nhiều thế hệ.

Ví dụ minh họa: Xét bài toán Tìm mật khẩu, yêu cầu của bài toán như sau:

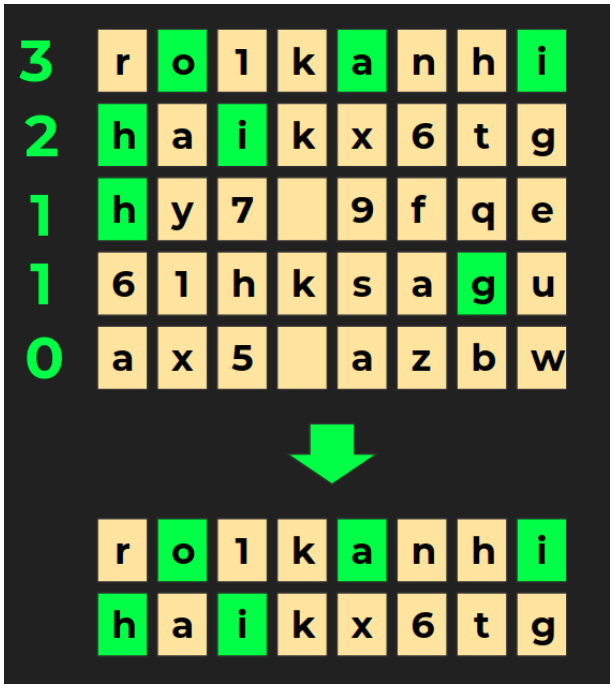
- Mật khẩu gồm 8 kí tự (bao gồm chữ cái, chữ số và khoảng trắng) - Ví dụ: hoilamgi.
- Mỗi lần thử, hệ thống sẽ báo về số lượng kí tự đúng với mật khẩu.
- Yêu cầu tìm ra chuỗi mật khẩu cho trước.



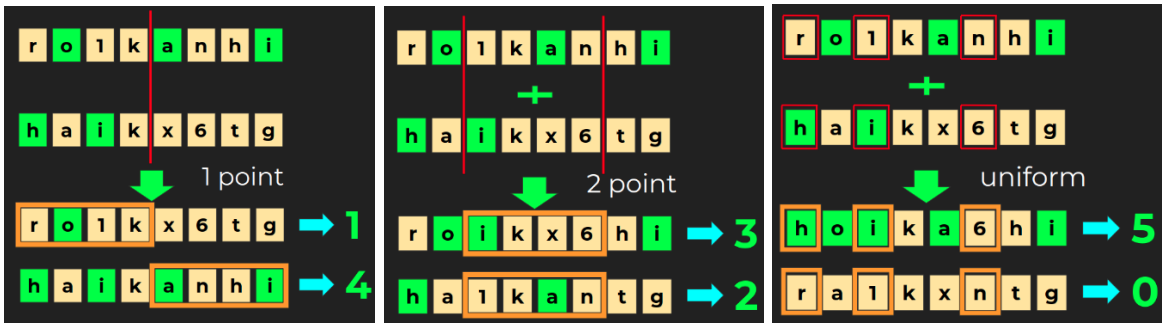
Hình 1: Khởi tạo quần thể bất kỳ với chiều dài mỗi mật khẩu là 8 ký tự



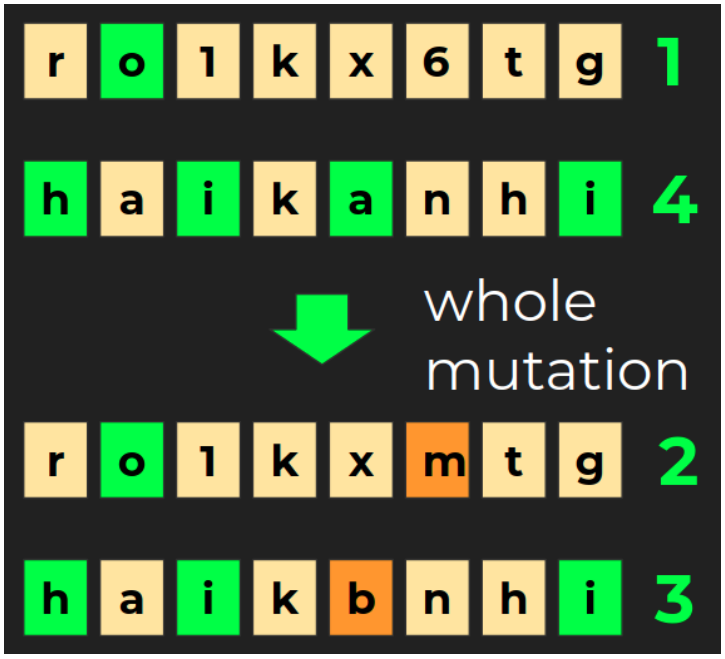
Hình 2: Đánh giá mật khẩu trong quần thể



Hình 3: lựa chọn cặp bố mẹ khỏe nhất sau khi đánh giá



Hình 4 a,b,c: Lai ghép cặp bố mẹ rồi đánh giá cặp con tương ứng



Hình 5: Đột biến con lai tạo sau khi lai ghép

2.1.5 Các biến thể của GA

1. Steady-State Genetic Algorithm (SSGA)
    - Đặc điểm: Trong SSGA, một hoặc một số cá thể mới được tạo ra và ngay lập tức thay thế một số cá thể kém thích nghi nhất trong quần thể, thay vì thay thế toàn bộ quần thể như trong GA tiêu chuẩn.
    - Ưu điểm: Giúp duy trì sự ổn định của quần thể và thường nhanh chóng hội tụ hơn so với GA tiêu chuẩn.
2. Elitist Genetic Algorithm
    - Đặc điểm: Phương pháp này bảo tồn một hoặc một số cá thể tốt nhất trong mỗi thế hệ (elites) không bị thay thế trong quá trình sinh sản.
    - Ưu điểm: Ngăn chặn mất mát của các giải pháp tốt nhất đã tìm thấy, đảm bảo rằng hiệu suất của quần thể không giảm qua các thế hệ.
3. Multi-Objective Genetic Algorithms (MOGA)
    - Đặc điểm: Được thiết kế để giải quyết các bài toán có nhiều mục tiêu tối ưu hóa đồng thời, mà không cần chuyển đổi chúng thành một mục tiêu đơn lẻ.
    - Ưu điểm: Có khả năng tìm ra một tập hợp các giải pháp tối ưu đa dạng, mỗi giải pháp tốt ở một mục tiêu nhất định, gọi là Pareto front.
4. Adaptive Genetic Algorithms
    - Đặc điểm: Tự động điều chỉnh các tham số của GA (như tỷ lệ đột biến, tỷ lệ lai ghép) dựa trên tiến trình tối ưu hóa.
    - Ưu điểm: Tăng khả năng thích ứng với bài toán cụ thể, cải thiện hiệu quả tìm kiếm và hội tụ.

5. Hybrid Genetic Algorithms
    - Đặc điểm: Kết hợp GA với các kỹ thuật tối ưu hóa khác (như Hill Climbing, Simulated Annealing, hoặc thậm chí là các thuật toán khác như Particle Swarm Optimization).
    - Ưu điểm: Tận dụng điểm mạnh của từng phương pháp để cải thiện hiệu quả tìm kiếm và khả năng hội tụ, đặc biệt hữu ích cho các bài toán phức tạp.
6. Parallel Genetic Algorithms
    - Đặc điểm: Phân tán quá trình GA trên nhiều máy tính hoặc nhiều luồng xử lý để thực hiện đồng thời.
    - Ưu điểm: Giảm thời gian tính toán đáng kể, đặc biệt quan trọng cho các bài toán lớn.
7. Real-Coded Genetic Algorithms
    - Đặc điểm: Sử dụng biểu diễn số thực thay vì chuỗi nhị phân để mã hóa các biến số.
    - Ưu điểm: Cải thiện độ chính xác và hiệu quả của GA trong các bài toán có biến số liên tục.
8. Constraint-Handling Techniques in GA
    - Đặc điểm: Áp dụng các kỹ thuật đặc biệt để xử lý các ràng buộc trong bài toán, giúp GA có thể áp dụng cho các bài toán tối ưu hóa có ràng buộc.
    - Ưu điểm: Mở rộng khả năng áp dụng của GA cho một loạt các bài toán thực tế có các ràng buộc phức tạp.

2.2 Thuật toán tối ưu hóa đàn kiến

- ACO được phát triển bởi Marco Dorigo vào đầu những năm 1990 trong khuôn khổ luận án tiến sĩ của ông tại Đại học Libre de Bruxelles, Bỉ. Thuật toán ban đầu được thiết kế để giải quyết bài toán người bán hàng (Travelling Salesman Problem - TSP), một trong những bài toán tối ưu hóa tổ hợp phổ biến và thách thức.
- Thuật toán ACO lấy cảm hứng từ hành vi tìm đường và xây dựng tổ của loài kiến trong tự nhiên. Khi kiếm ăn, kiến phát tán một chất hóa học gọi là pheromone trên đường đi của chúng. Các con kiến khác có thể cảm nhận được mùi pheromone và theo dõi nó để tìm thức ăn. Đường đi có nhiều pheromone hơn (tức là được nhiều kiến đi qua) trở nên hấp dẫn hơn, dẫn đến một quá trình tối ưu hóa tự nhiên qua đó đường đi tốt nhất dần được thiết lập.

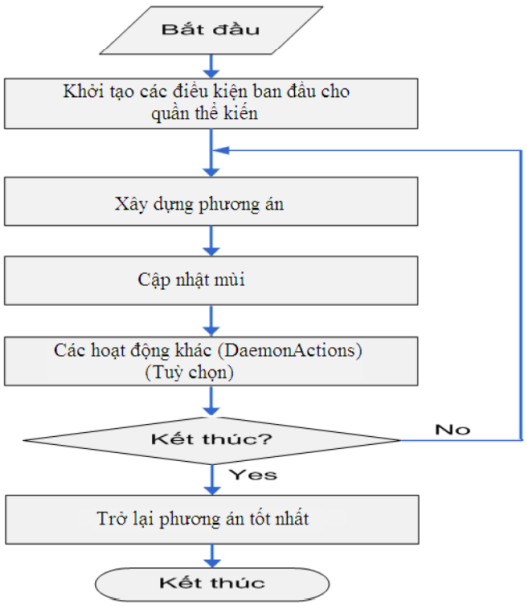
- 

2.1.1Các thành phần chính của ACO

1. Kiến (Ants)
    - Mô tả: Trong ACO, kiến được mô phỏng là các tác nhân giải quyết bài toán. Chúng bắt đầu từ một điểm, thường là điểm khởi đầu của bài toán, và dần dần xây dựng lộ trình bằng cách lựa chọn từng bước tiếp theo dựa trên pheromone và thông tin heuristic.

- Vai trò: Tác nhân của thuật toán, mỗi con kiến tương ứng với một lần chạy thuật toán hoặc một lần thử nghiệm giải pháp, và chúng cộng tác để khám phá không gian giải pháp.

2. Pheromone
   - Mô tả: Pheromone trong ACO là một biến số ảo mà kiến "để lại" trên lộ trình mà chúng đã đi qua. Lượng pheromone trên một lộ trình phản ánh mức độ thành công của lộ trình đó trong quá khứ, càng nhiều kiến đi qua và càng thành công thì lượng pheromone càng cao.
   - Vai trò: Đây là cơ chế chính để chia sẻ thông tin giữa các kiến, giúp thuật toán tập trung vào những lộ trình tốt hơn và tránh những lộ trình kém hiệu quả.
3. Heuristic Information
   - Mô tả: Thông tin heuristic là dữ liệu dựa trên bản chất của bài toán, chẳng hạn như khoảng cách giữa các điểm trong bài toán người bán hàng, hoặc chi phí liên quan đến một bước chuyển động trong bài toán tối ưu hóa.
   - Vai trò: Hỗ trợ kiến trong việc lựa chọn bước tiếp theo bằng cách cung cấp một "mức độ ưu tiên" tự nhiên dựa trên các tính toán đơn giản, giúp kết hợp sự khám phá ngẫu nhiên với việc tập trung vào giải pháp hiệu quả.
4. Hàm Mục tiêu (Objective Function)
   - Mô tả: Hàm mục tiêu là tiêu chuẩn để đánh giá chất lượng của một lộ trình hoặc giải pháp. Trong ACO, hàm mục tiêu thường được dùng để tính toán "mức độ thích hợp" của một lộ trình dựa trên tổng chi phí, khoảng cách, thời gian, hoặc các yếu tố tương tự.
   - Vai trò: Xác định mức độ cập nhật pheromone trên các lộ trình; lộ trình càng tối ưu theo hàm mục tiêu, lượng pheromone để lại càng nhiều, từ đó hướng dẫn các kiến trong các lần tìm kiếm sau tập trung vào những lộ trình đó.



```
Procedure Thuật toán ACO;
Begin
   Khởi tạo tham số, ma trận mùi, khởi tạo m con kiến;
   repeat
      for k = 1 to m do
         Kiến k xây dựng lời giải;
      end-for
      Cập nhật mùi;
      Cập nhật lời giải tốt nhất;
   until (Điều kiện kết thúc);
   Đưa ra lời giải tốt nhất;
End;
```

Hình 7 Mã giả

2.2.2 Các tính chất của ACO

1) **Tính chính xác (Accuracy):** ACO đạt được tính chính xác thông qua việc cập nhật đường đi tốt nhất dựa trên lượng pheromone mà các "kiến" để lại trên đường đi của chúng. Khi nhiều kiến lựa chọn một đường đi và để lại pheromone, đường đi đó trở nên hấp dẫn hơn, dẫn đến khả năng cao mà đường đó là lối đi tối ưu hoặc gần tối ưu. Tuy nhiên, ACO không đảm bảo tìm được giải pháp tối ưu toàn cục 100% trong mọi trường hợp, nhưng nó thường tìm được giải pháp rất tốt trong thực tế.
2) **Tính khách quan (Objectivity):** ACO hoạt động dựa trên quy tắc khách quan là mức độ tích lũy pheromone, không phụ thuộc vào bất kỳ yếu tố chủ quan nào khác từ bên ngoài. Tuy nhiên, cách thiết lập tham số ban đầu và cập nhật pheromone có thể ảnh hưởng đến kết quả, nhưng nhìn chung, quá trình tìm kiếm giải pháp của ACO là khách quan vì nó dựa trên thông tin mà kiến thu thập được.
3) **Tính phổ dụng (Generality):** ACO được áp dụng rộng rãi cho nhiều loại bài toán tối ưu hóa, từ bài toán người du lịch (TSP) đến các bài toán lập lịch và phân bổ tài nguyên. Sự linh hoạt này là do cơ chế tự thích ứng của ACO, cho phép nó tìm kiếm giải pháp hiệu quả trong nhiều không gian tìm kiếm khác nhau.
4) **Tính rõ ràng (Clarity):** Mặc dù ACO là một thuật toán phức tạp, các bước của nó khá rõ ràng: khởi tạo pheromone, kiến di chuyển dựa trên pheromone và hàm giá trị, cập nhật pheromone dựa trên chất lượng của lối đi, và lặp lại quá trình. Cách tiếp cận này là minh bạch và có thể được theo dõi dễ dàng trong các bước triển khai.
5) **Tính kết thúc (Termination):** ACO thường sử dụng một số tiêu chí dừng cụ thể, như số lần lặp tối đa, không có cải thiện trong chất lượng đường đi sau nhiều lần lặp, hoặc đạt được một giải pháp có độ chính xác nhất định. Điều này giúp đảm bảo rằng thuật toán không chạy vô hạn và tài nguyên tính toán được sử dụng một cách hiệu quả.

2.2.3 Độ phức tạp

Độ phức tạp của Thuật toán Tối ưu Hóa Bầy Kiến (Ant Colony Optimization - ACO) có thể khá phức tạp để phân tích chính xác vì nó phụ thuộc vào nhiều yếu tố như kích thước của bài toán, số lượng kiến trong mô phỏng, và cách thức cập nhật pheromone.

Độ phức tạp thời gian

Số lượng kiến (m): Mỗi con kiến trong quần thể tạo một lộ trình hoàn chỉnh trong bài toán.

Số thành phần (n): Đây có thể là số lượng thành phố trong TSP. Mỗi kiến cần xem xét mỗi thành phần để xây dựng lộ trình.

Số lần lặp (t): Đại diện cho số lần thuật toán thực hiện quá trình tìm kiếm giải pháp, bao gồm cả việc cập nhật pheromone.

Chi phí tính toán cho mỗi lượt di chuyển của kiến (c): Đây là chi phí để mỗi con kiến tạo ra một lộ trình hoàn chỉnh. Trong trường hợp tồi tệ nhất, chi phí này có thể là $O(n^2)$, khi mỗi con kiến xem xét mọi cặp thành phần có thể.

Độ phức tạp thời gian tổng thể của ACO, trong trường hợp tồi tệ nhất, khi xem xét tất cả các yếu tố trên, có thể là:

$$O(t \times m \times n^2)$$

Trong đó t là số lần lặp, m là số lượng kiến, và đại diện cho chi phí tạo một lộ trình hoàn chỉnh của một kiến, giả sử mỗi kiến cần xem xét mọi cặp thành phần có thể. Đây là ước lượng cho trường hợp phức tạp nhất, trong thực tế, tùy vào thiết kế cụ thể của thuật toán và các cải tiến được áp dụng, độ phức tạp có thể thấp hơn.

Độ phức tạp không gian

ACO cũng yêu cầu một lượng lớn bộ nhớ để lưu trữ thông tin pheromone trên mọi cạnh của đồ thị, điều này có độ phức tạp không gian là:

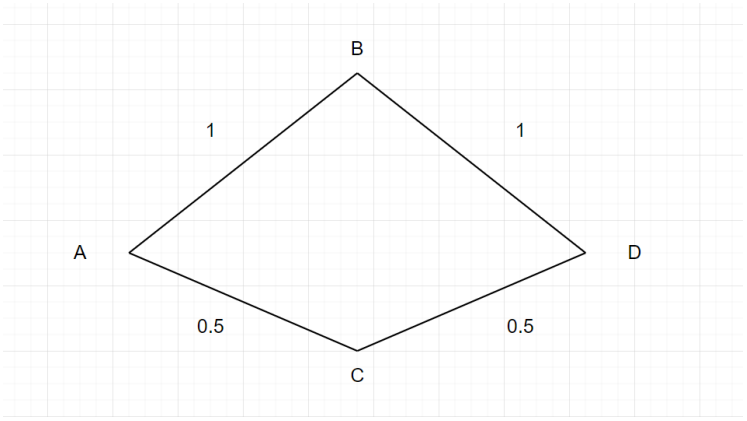
$$O(n^2)$$

đối với bài toán như TSP, nơi pheromone cần được lưu trữ cho mỗi cặp thành phố.

2.2.4 Bài toán điển hình

Bài toán tìm đường đi ngắn nhất giữa điểm bắt đầu tới điểm đích là 1 bài toán điển hình của ACO nhằm tối ưu đường đi tới đích một cách ngắn nhất nhanh nhất.

Ví dụ: Cho 4 đỉnh A, B, C, D với AB=BD = 1 , AC=CD=0.5 đơn vị khoảng cách tổ kiến bắt đầu từ A và thức ăn tại D. Tìm đường đi ngắn nhất



Khởi tạo các giá trị

- $\tau_{A,B}, \tau_{A,C}$: Mức pheromone ban đầu trên các cạnh A-B và A-C. Giả sử ban đầu là 0.1 cho cả hai cạnh
- $\eta_{A,B}, \eta_{A,C}$: Độ thuận lợi của cạnh, tính bằng nghịch đảo của khoảng cách từ A đến B và từ A đến C. với khảng cách AB = 1 và AC = 0.5, ta có $\eta_{A,B}$ = 1/1 = 1 và $\eta_{A,C}$ = 1/0.5 = 2
- α, β: Các tham số điều chỉnh ảnh hưởng của pheromone (α) và thông tin heuristic (β). Giả sử $\alpha = 1,\ \beta = 2$.

Công thức tính xác suất:

Xác suất $P_{i,j}$ để kiến chọn đi từ điểm i đến j được tính bằng công thức:

$$P_{i,j} = \frac{(\tau_{i,j})^{\alpha}.(\eta_{i,j})^{\beta}}{\sum_{k \in adj(i)}(\tau_{i,k})^{\alpha}.(\eta_{i,k})^{\beta}}$$

Xác suất từ A đến B ($P_{A,B}$):

$$P_{A,B} = \frac{(0.1)^1.(1)^2}{(0.1)^1.(1)^2 + (0.1)^1.(2)^2} = \frac{0.1}{0.5} = 0.2$$

Giải thích:

- $\tau_{A,B}, \tau_{A,C} = 0.1$
- $\eta_{A,B} = 1, \eta_{A,C} = 2$
- $\alpha = 1, \beta = 2$

Xác suất từ A đến B ($P_{A,B}$):

$$P_{A,C} = \frac{(0.1)^1.(2)^2}{(0.1)^1.(1)^2 + (0.1)^1.(2)^2} = \frac{0.4}{0.5} = 0.8$$

=> Từ đó kiến sẽ chọn đường đi có xác suất cao nhất để đi. Nhưng vẫn còn có trong số các con kiến đi đường có xác suất ít để đi và kiểm tra đường đi khác, vì đâu chắc tổng đường đi chứa đường có xác suất nhỏ nhất đang xét nó nhỏ hơn tổng đường đi của các đường chứa đường có xác xuất lớn hơn.

2.2.5 Các biến thể của ACO

1. Ant System (AS)
    - Mô tả: AS là phiên bản ban đầu của ACO. Trong AS, tất cả kiến được cho là cập nhật pheromone dựa trên chất lượng của lộ trình mà chúng đã tìm thấy.
    - Ứng dụng: Phù hợp cho các bài toán có không gian tìm kiếm không quá lớn, nơi mà sự thăm dò có thể được thực hiện một cách có hệ thống.
2. Ant Colony System (ACS)
    - Mô tả: Trong ACS, chỉ những kiến tìm thấy lộ trình tốt nhất mới được cập nhật pheromone, điều này giúp tập trung vào việc khai thác các lộ trình tối ưu.
    - Ứng dụng: Đặc biệt hiệu quả trong các bài toán đòi hỏi khả năng khai thác nhanh chóng các giải pháp tốt.
3. Max-Min Ant System (MMAS)
    - Mô tả: MMAS đặt giới hạn trên và dưới cho lượng pheromone, ngăn chặn sự thống trị của bất kỳ lộ trình nào để đảm bảo sự khám phá đầy đủ hơn của không gian tìm kiếm.
    - Ứng dụng: Phù hợp cho các bài toán có nguy cơ cao bị mắc kẹt ở cực tiểu địa phương.
4. Ant-Q
    - Mô tả: Là sự kết hợp giữa ACO và Q-learning, một kỹ thuật học tăng cường. Trong Ant-Q, kiến cập nhật giá trị của pheromone dựa trên một quá trình học có thưởng thức.
    - Ứng dụng: Hiệu quả trong các môi trường động, nơi các kiến cần thích ứng với sự thay đổi của bài toán theo thời gian.
5. Rank-based Ant System (RAS)
    - Mô tả: Chỉ những kiến tìm thấy các lộ trình tốt nhất mới được phép cập nhật pheromone, nhưng dựa trên xếp hạng thay vì chỉ có một kiến tốt nhất.
    - Ứng dụng: Cung cấp sự cân bằng tốt hơn giữa khám phá và khai thác, thích hợp cho các bài toán có không gian tìm kiếm rộng.
6. Hypercube Framework for Ant Colony Optimization (HACO)
    - Mô tả: Trong HACO, pheromone được xử lý trong một không gian nhiều chiều, cho phép thể hiện phức tạp hơn các mối quan hệ giữa các thành phần.
    - Ứng dụng: Đặc biệt hữu ích trong các bài toán tối ưu hóa đa mục tiêu hoặc khi các quyết định có mối liên hệ phức tạp với nhau.

Phần 3: Ứng dụng GA và ACO vào bài toán người bán hàng (Travelling Salesman Problem - TSP)

3.1 Ý tưởng thực hiện

3.1.1 Thuật toán Di truyền (Genetic Algorithms - GA)

Ý tưởng cơ bản: Mô phỏng quá trình tiến hóa sinh học, bao gồm các phép toán như chọn lọc tự nhiên, lai ghép và đột biến để tìm ra giải pháp tối ưu.

Cách tiếp cận:

- Khởi tạo: Tạo một quần thể ban đầu gồm nhiều cá thể, mỗi cá thể là một giải pháp tiềm năng (lộ trình qua các thành phố).
- Đánh giá: Tính toán độ thích nghi của mỗi cá thể, thường là tổng khoảng cách của lộ trình.
- Chọn lọc: Chọn các cá thể có độ thích nghi cao để sinh sản.
- Lai ghép và Đột biến: Tạo ra thế hệ mới thông qua các phép lai ghép (kết hợp đặc điểm của hai cá thể cha mẹ) và đột biến (thay đổi ngẫu nhiên một phần của giải pháp).

Lặp lại: Quá trình này được lặp đi lặp lại cho đến khi đạt được giải pháp mong muốn hoặc khi đã thực hiện đủ số thế hệ.

3.1.2 Thuật toán tối ưu hóa đàn kiến (Ant Colony Optimization - ACO)

Ý tưởng cơ bản: Mô phỏng hành vi tìm đường và xây dựng đường đi của kiến trong tự nhiên, dựa trên việc tích lũy pheromone trên đường đi để hướng đến giải pháp tối ưu.

Cách tiếp cận:

- Khởi tạo: Đặt một lượng pheromone ban đầu nhỏ trên tất cả các cạnh nối giữa các thành phố.
- Xây dựng lộ trình: Mỗi con kiến sẽ chọn lộ trình tiếp theo dựa trên xác suất, ưu tiên các cạnh có nhiều pheromone và khoảng cách ngắn.
- Cập nhật Pheromone: Sau khi tất cả kiến hoàn thành lộ trình của mình, giảm pheromone trên tất cả các cạnh và tăng cường pheromone trên lộ trình tối ưu nhất mà kiến đã đi qua.

Lặp lại: Quá trình này được lặp lại cho đến khi đạt đến một điều kiện dừng, chẳng hạn như số lần lặp tối đa hoặc khi giải pháp không còn cải thiện nữa.

3.2 Thực nghiệm

Tạo dữ liệu thành phố và khoảng cách giữa các thành phố

```python
def generate_random_cities(n_cities):
    np.random.seed(0)  # Để kết quả có thể lặp lại
    cities = np.random.rand(n_cities, 2) * 100  # Tạo ngẫu nhiên tọa độ cho các thành phố trong một phạm vi 0-100
    distances = np.zeros((n_cities, n_cities))  # Khởi tạo ma trận khoảng cách
    city_names = [f"City {i}" for i in range(n_cities)]  # Tạo tên các thành phố

    # Tính toán và làm tròn khoảng cách giữa các thành phố
    for i in range(n_cities):
        for j in range(n_cities):
            distance = np.linalg.norm(cities[i] - cities[j])
            distances[i][j] = np.round(distance)

    return distances, city_names, cities
```

3.2.1 Thuật toán Di truyền (Genetic Algorithms - GA)

Code:

1. Hàm khởi tạo

```python
def __init__(self, distances, n_population=10, n_generations=100, mutation_rate=0.01):
    # Khởi tạo các thuộc tính của thuật toán di truyền
    self.distances = distances  # Ma trận khoảng cách giữa các thành phố
    self.n_population = n_population  # Số lượng cá thể trong quần thể
    self.n_generations = n_generations  # Số thế hệ tiến hóa
    self.mutation_rate = mutation_rate  # Tỷ lệ đột biến
    # Tạo ngẫu nhiên các cá thể trong quần thể ban đầu
    self.population = [np.random.permutation(len(distances)) for _ in range(n_population)]
```

2. Hàm tính độ thích nghi của 1 cá thể

```python
def fitness(self, chromosome):
    # Hàm tính độ thích nghi của một cá thể (độ dài của đường đi)
    return sum([self.distances[chromosome[i], chromosome[i + 1]] for i in range(-1, len(chromosome) - 1)])
```

3. Hàm chọn cá thể

```python
def select(self, population):
    # Tính toán fitness cho mỗi cá thể trong quần thể. Fitness càng thấp (tổng khoảng cách càng ngắn) càng tốt.
    fitnesses = np.array([self.fitness(chromosome) for chromosome in population])

    # Chuyển đổi fitness thành một giá trị mà ở đó cá thể có tổng khoảng cách ngắn nhất có giá trị fitness cao nhất
    fitnesses = np.max(fitnesses) - fitnesses

    # Kiểm tra nếu tất cả các cá thể có cùng fitness (tức là tất cả các khoảng cách tổng là như nhau)
    if np.sum(fitnesses) == 0:
        # Nếu tất cả các fitness là bằng nhau (không có sự khác biệt), chọn một cá thể ngẫu nhiên
        return population[np.random.randint(len(population))]

    # Chuẩn hóa các giá trị fitness để chúng cộng lại thành 1, tạo thành một phân phối xác suất
    fitnesses = fitnesses / np.sum(fitnesses)

    # Chọn một cá thể từ quần thể dựa trên phân phối xác suất đã tính, cá thể càng có fitness cao càng có xác suất cao được chọn
    idx = np.random.choice(np.arange(len(population)), p=fitnesses)

    # Trả về cá thể đã chọn
    return population[idx]
```

4. Hàm lai chéo

```
def crossover(self, parent1, parent2):
    # Hàm lai ghép giữa hai cá thể để tạo ra các cá thể con mới
    size = len(parent1)
    c1, c2 = np.zeros(size, dtype=int), np.zeros(size, dtype=int)
    cut1, cut2 = sorted(random.sample(range(size), 2))
    c1_in, c2_in = parent1[cut1:cut2], parent2[cut1:cut2]
    c1[cut1:cut2], c2[cut1:cut2] = c1_in, c2_in
    fill_pos = list(range(cut1)) + list(range(cut2, size))
    for i in fill_pos:
        for j in range(size):
            if parent2[j] not in c1:
                c1[i] = parent2[j]
                break
        for j in range(size):
            if parent1[j] not in c2:
                c2[i] = parent1[j]
                break
    return [c1, c2]
```

5. Hàm đột biến cá thể

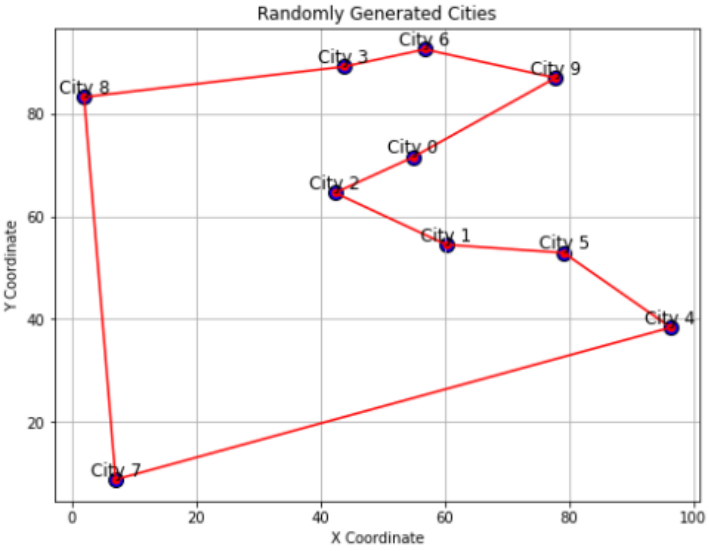
```
def crossover(self, parent1, parent2):
    # Hàm lai ghép giữa hai cá thể để tạo ra các cá thể con mới
    size = len(parent1)
    c1, c2 = np.zeros(size, dtype=int), np.zeros(size, dtype=int)
    cut1, cut2 = sorted(random.sample(range(size), 2))
    c1_in, c2_in = parent1[cut1:cut2], parent2[cut1:cut2]
    c1[cut1:cut2], c2[cut1:cut2] = c1_in, c2_in
    fill_pos = list(range(cut1)) + list(range(cut2, size))
    for i in fill_pos:
        for j in range(size):
            if parent2[j] not in c1:
                c1[i] = parent2[j]
                break
        for j in range(size):
            if parent1[j] not in c2:
                c2[i] = parent1[j]
                break
    return [c1, c2]
```

6. Hàm chạy thuật toán

```
def run(self):
    # Hàm chạy thuật toán di truyền qua các thế hệ
    for _ in range(self.n_generations):
        new_population = []
        for _ in range(self.n_population // 2):
            # Chọn lọc và lai ghép để tạo ra thế hệ mới
            parent1, parent2 = self.select(self.population), self.select(self.population)
            child1, child2 = self.crossover(parent1, parent2)
            new_population.extend([self.mutate(child1), self.mutate(child2)])
        self.population = new_population
    # Trả về cá thể có đường đi ngắn nhất trong quần thể cuối cùng
    return min(self.population, key=self.fitness)
```

Kết quả với 10 thành phố:

- Best route: [9 6 3 8 7 4 5 1 2 0]
- Best distance: 351.0
- Execution time: 0.18 seconds



3.2.2 Thuật toán tối ưu hóa đàn kiến (Ant Colony Optimization - ACO)

Code:

7. Hàm khởi tạo

```
def __init__(self, distances, city_names, n_ants, n_best, n_iterations, decay, alpha=1, beta=1):
    self.distances = distances
    self.city_names = city_names
    self.pheromone = np.ones(self.distances.shape) / len(distances)
    self.all_inds = range(len(distances))
    self.n_ants = n_ants
    self.n_best = n_best
    self.n_iterations = n_iterations
    self.decay = decay
    self.alpha = alpha
    self.beta = beta
```

8. Hàm cập nhật pheromone cho các đường đi

```
def spread_pheromone(self, all_paths, n_best, shortest_path):
    # Sắp xếp tất cả các đường đi dựa trên độ dài (khoảng cách) của chúng
    sorted_paths = sorted(all_paths, key=lambda x: x[1])

    # Lặp qua n_best đường đi ngắn nhất để tăng cường pheromone
    for path, dist in sorted_paths[:n_best]:
        # Tăng cường pheromone trên mỗi bước di chuyển trong đường đi
        for move in path:
            # Cập nhật lượng pheromone trên từng đoạn đường dựa vào độ dài đoạn đường đó
            # Bằng cách lấy nghịch đảo của khoảng cách: pheromone càng cao khi khoảng cách càng ngắn
            self.pheromone[move] += 1.0 / self.distances[move]
```

9. **Hàm sinh ra đường đi và sinh ra tất cả đường đi, cùng với hàm tính quãng đường**

```
def gen_path(self, start):
    # Khởi tạo danh sách path để lưu trữ đường đi
    path = []
    # Tạo một tập hợp visited để theo dõi các đỉnh đã thăm
    visited = set()
    # Đánh dấu đỉnh bắt đầu là đã thăm
    visited.add(start)
    # Đặt đỉnh hiện tại là đỉnh bắt đầu
    prev = start

    # Lặp cho đến khi đường đi chưa đạt đủ số đỉnh (trừ điểm bắt đầu)
    while len(path) < len(self.distances) - 1:
        # Chọn nước đi tiếp theo dựa trên pheromone và khoảng cách, tránh các đỉnh đã thăm
        move = self.pick_move(self.pheromone[prev], self.distances[prev], visited)
        # Thêm nước đi vào đường đi
        path.append((prev, move))
        # Cập nhật đỉnh hiện tại là đỉnh vừa di chuyển tới
        prev = move
        # Đánh dấu đỉnh mới là đã thăm
        visited.add(move)

    # Kết thúc đường đi bằng cách quay trở lại điểm bắt đầu
    path.append((prev, start))
    # Trả về đường đi hoàn chỉnh
    return path
```

```
def gen_all_paths(self):
    # Khởi tạo một danh sách rỗng để lưu trữ tất cả các đường đi được sinh ra
    all_paths = []

    # Lặp qua số lượng kiến đã được định trước (n_ants)
    for i in range(self.n_ants):
        # Sinh ra một đường đi cho mỗi con kiến từ điểm bắt đầu (ví dụ: điểm 0)
        path = self.gen_path(0)

        # Tính toán khoảng cách tổng của đường đi vừa sinh ra
        path_distance = self.gen_path_dist(path)

        # Thêm cặp (đường đi, khoảng cách) vào danh sách các đường đi
        all_paths.append((path, path_distance))

    # Trả về danh sách chứa tất cả các đường đi và khoảng cách tương ứng của chúng
    return all_paths
```

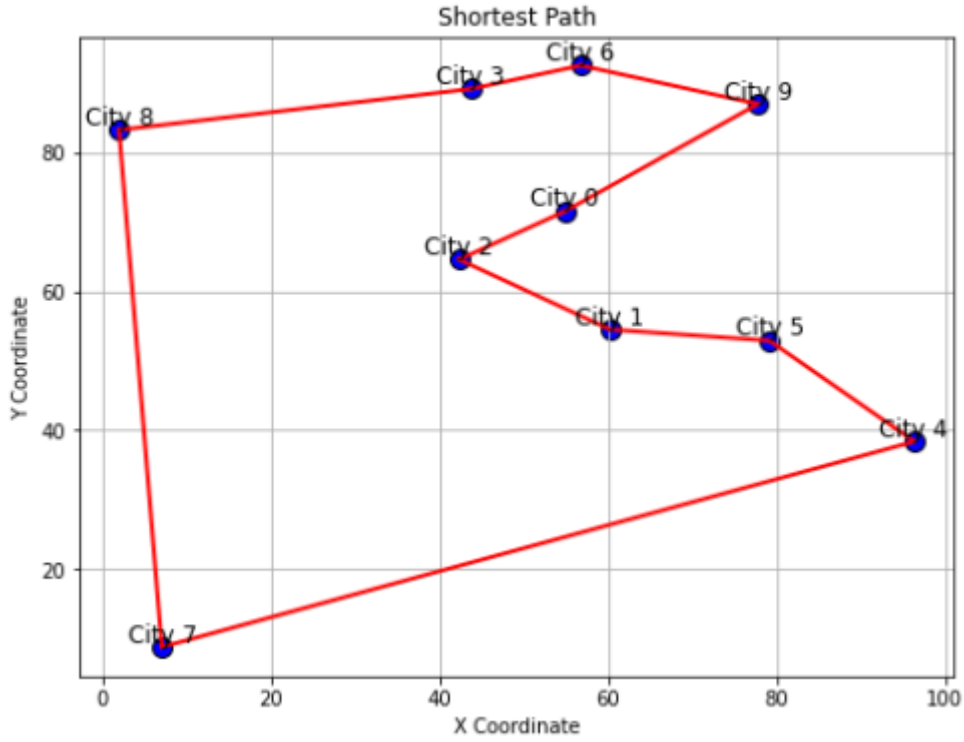
```
def gen_path_dist(self, path):
    total_dist = 0
    for (start, end) in path:
        total_dist += self.distances[start][end]
    return total_dist
```

10. Hàm di chuyển cho kiến

```python
def pick_move(self, pheromone, distances, visited):
    # Sao chép mảng pheromone để không làm thay đổi mảng ban đầu
    pheromone = np.copy(pheromone)
    # Đặt giá trị pheromone của các điểm đã thăm bằng 0 để tránh lặp lại
    pheromone[list(visited)] = 0

    # Tạo một heuristic dựa trên nghịch đảo khoảng cách (vô cùng nếu khoảng cách là 0)
    heuristic = 1.0 / np.where(distances == 0, np.inf, distances)

    # Tính toán giá trị cho mỗi lựa chọn di chuyển dựa trên pheromone và heuristic
    row = pheromone ** self.alpha * (heuristic ** self.beta)

    # Nếu tổng giá trị của các lựa chọn di chuyển là 0, tức là không còn lựa chọn hợp lệ
    if np.sum(row) == 0:
        # Tạo danh sách các chỉ số có thể di chuyển, loại trừ các điểm đã thăm và khoảng cách là 0
        available = [i for i in self.all_inds if i not in visited and distances[i] > 0]

        # Nếu không còn điểm nào có thể di chuyển, trả về điểm bắt đầu (cần định nghĩa biến start)
        if not available:
            return start
        # Ngược lại, chọn ngẫu nhiên một điểm có thể di chuyển từ danh sách available
        return np.random.choice(available)
    else:
        # Chuẩn hóa các giá trị di chuyển để tổng là 1, tạo thành xác suất
        norm_row = row / np.sum(row)
        # Chọn một điểm di chuyển dựa trên xác suất đã tính
        move = np.random.choice(self.all_inds, 1, p=norm_row)[0]
        return move
```

Kết quả với 10 thành phố:

- Shortest path: ([(0, 2), (2, 1), (1, 5), (5, 4), (4, 7), (7, 8), (8, 3), (3, 6), (6, 9), (9, 0)])
- Best distance: 351.0
- Execution time: 0.33 seconds



3.3 Kết luận và đánh giá

3.3.1 Đánh giá hiệu quả

Thuật toán di truyền (GA)

Ưu điểm: GA tốt trong việc tìm kiếm trên không gian lớn và có khả năng thoát khỏi các điểm tối ưu cục bộ nhờ vào cơ chế đột biến và lai ghép.

Nhược điểm: Kết quả tối ưu phụ thuộc nhiều vào các tham số như tỷ lệ đột biến và cách thức lai ghép. Đôi khi, việc tinh chỉnh các tham số này để đạt hiệu quả tối ưu có thể rất phức tạp và mất thời gian.

Thuật toán kiến (ACO)

Ưu điểm: ACO rất mạnh trong việc tìm kiếm đường đi dựa trên cơ chế pheromone, cho phép chia sẻ thông tin giữa các cá thể, từ đó hướng tới giải pháp tốt chung.

Nhược điểm: ACO có thể mất nhiều thời gian hơn để hội tụ, đặc biệt là khi kích thước bài toán lớn, do nó phụ thuộc vào sự phân bố pheromone và cần nhiều vòng lặp để điều chỉnh pheromone đạt được mức tối ưu.

3.3.2 Phức tạp thuật toán

- GA: Phức tạp chủ yếu ở các bước lai ghép và đột biến. Việc quản lý quần thể đòi hỏi kỹ thuật lập trình tốt để đảm bảo hiệu quả.
- ACO: Phức tạp ở phần tính toán pheromone và cách thức các kiến chọn lựa đường đi. Đòi hỏi sự cân bằng giữa các tham số như suy giảm pheromone và hệ số heuristic.

3.3.3 Tính mềm dẻo và khả năng thích ứng

- GA: Có khả năng thích ứng cao với các bài toán khác nhau bằng cách thay đổi cách thức lai ghép và đột biến.
- ACO: Tuy nhiên, ACO cũng có thể thích ứng với nhiều loại bài toán tối ưu hóa khác nhau, nhưng cần điều chỉnh cẩn thận các tham số liên quan đến pheromone.

3.3.4 Tính chất hội tụ

- GA: Có thể hội tụ nhanh chóng nếu quần thể đa dạng và kích thước quần thể đủ lớn.
- ACO: Hội tụ dựa trên sự tích lũy và bay hơi pheromone, có thể chậm hơn GA nhưng đôi khi lại mang lại giải pháp tốt hơn do tính toán kỹ lưỡng hơn.

Tài liệu tham khảo

1 Tài liệu chuyên tin học quyển 3

1 https://www.geeksforgeeks.org/genetic-algorithms/?ref=header_search
2 https://www.geeksforgeeks.org/encoding-methods-in-genetic-algorithm/?ref=header_search
3 https://www.geeksforgeeks.org/project-idea-genetic-algorithms-for-graph-colouring/?ref=header_search'
4 https://intapi.sciendo.com/pdf/10.2478/v10238-012-0039-2
5 https://core.ac.uk/download/pdf/32452919.pdf
6 https://www.linkedin.com/pulse/use-genetic-algorithms-planning-scheduling-projects-well-suleiman
7 https://nerophung.github.io/2020/05/28/genetic-algorithm#thu%E1%BA%ADt-to%C3%A1n-di-truy%E1%BB%81n
8 https://medium.datadriveninvestor.com/genetic-algorithm-made-intuitive-with-natural-selection-and-python-project-from-scratch-3462f7793a3f
9 https://www.youtube.com/watch?v=fxCHAkYAjis&list=PLALZwazFybKKeKPXDP4EjrDLNW436xiLI&index=2
10 https://www.youtube.com/watch?v=MGY3Vf_M4Qs
11 https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
12 https://www.geeksforgeeks.org/introduction-to-ant-colony-optimization/
13 https://www.geeksforgeeks.org/introduction-to-ant-colony-optimization/?ref=header_search
14 https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3