**Đề bài: Nghiên cứu cơ sở lý thuyết, ứng dụng và cài đặt giải thuật di truyền để giải bài toán Người du lịch (Traveling Salesman program)**

**MỤC LỤC**

**LỜI MỞ ĐẦU**

Đề tài "Nghiên cứu cơ sở lý thuyết, ứng dụng và cài đặt giải thuật di truyền để giải bài toán Người du lịch (Traveling Salesman Problem - TSP)" nhằm tìm hiểu sâu sắc về bài toán TSP, một trong những bài toán nổi tiếng trong lĩnh vực tối ưu hóa và lập lịch. Bài toán này đặt ra yêu cầu tìm đường đi ngắn nhất cho một người du lịch để thăm tất cả các thành phố và trở về thành phố xuất phát. TSP có ứng dụng rộng rãi trong logistics, thiết kế mạng và nhiều lĩnh vực khác. Với sự phát triển nhanh chóng của công nghệ thông tin, nhu cầu tìm kiếm các giải pháp tối ưu cho bài toán TSP ngày càng gia tăng. Giải thuật di truyền, một trong những phương pháp tối ưu hóa heuristic mạnh mẽ, đã cho thấy tiềm năng vượt trội trong việc tìm ra các giải pháp gần đúng cho những bài toán phức tạp như TSP.

**I. GIỚI THIỆU ĐỀ TÀI**

1. Lý do chọn đề tài

Trong bối cảnh ngày nay, khi các vấn đề liên quan đến tối ưu hóa được ứng dụng rộng rãi trong nhiều lĩnh vực, bài toán Người du lịch (TSP) nổi lên như một bài toán kinh điển, có tính ứng dụng thực tiễn cao, đặc biệt trong ngành logistics và vận tải. Việc nghiên cứu và tìm lời giải cho bài toán này không chỉ mang ý nghĩa học thuật mà còn giúp giải quyết các thách thức thực tế.

Giải thuật di truyền (Genetic Algorithm) là một phương pháp mạnh mẽ để tìm kiếm lời giải gần đúng trong các bài toán phức tạp như TSP. Nghiên cứu này giúp khám phá tiềm năng của giải thuật di truyền trong việc tối ưu hóa các vấn đề thực tế, đặc biệt trong việc giải quyết bài toán NP-hard như TSP.

1. Mục tiêu báo cáo

Mục tiêu của đề tài này là nghiên cứu và ứng dụng giải thuật di truyền (Genetic Algorithm) để giải quyết bài toán Người du lịch (Traveling Salesman Problem - TSP), nhằm tối ưu hóa lộ trình di chuyển qua các thành phố với chi phí thấp nhất. Đề tài hướng đến việc giảm thiểu tổng quãng đường di chuyển, tiết kiệm chi phí và thời gian, từ đó góp phần cải thiện hiệu quả trong các lĩnh vực vận tải và logistics.

Ngoài ra, đề tài cũng tập trung vào việc đánh giá hiệu quả của giải thuật di truyền so với các phương pháp khác thông qua các tiêu chí như tốc độ hội tụ và chất lượng lời giải. Kết quả nghiên cứu không chỉ mang tính lý thuyết mà còn có khả năng ứng dụng thực tế, góp phần nâng cao hiệu quả hoạt động trong các bài toán tối ưu hóa liên quan.

Đồng thời, thông qua đề tài này, nhóm nghiên cứu mong muốn nâng cao năng lực nghiên cứu và ứng dụng các giải thuật hiện đại, góp phần đào tạo nguồn nhân lực chất lượng cao trong lĩnh vực khoa học máy tính, tối ưu hóa và logistics. Đề tài cũng mở ra hướng phát triển mới cho việc cải tiến giải thuật, nâng cao khả năng ứng dụng trong các bài toán tương tự trong tương lai.

1. Đối tượng và phạm vi nghiên cứu  
   3.1. Đối tượng nghiên cứu

Bài toán TSP và các biến thể của nó.  
3.2. Phạm vi nghiên cứu

- Giới hạn về giải thuật: Tập trung vào giải thuật di truyền, một trong những phương pháp tối ưu hóa heuristic phổ biến.

- Dữ liệu và mô hình: Sử dụng dữ liệu mô phỏng và thực tế từ các bài toán TSP điển hình để kiểm tra và đánh giá giải thuật.

1. Công cụ sử dụng

- Ngôn ngữ: Java.

- Công cụ sử dụng: Eclipse, NetBean, VS Code

**II. CƠ SỞ LÝ THUYẾT**

1. Một số khái niệm

Đồ thị (Graph) là một cặp G = (V, E), trong đó V là tập hợp các đỉnh (hoặc nút) và E là tập hợp các cạnh (hoặc liên kết) giữa các đỉnh trong V. Mỗi cạnh trong E có thể có hướng hoặc không hướng, và có thể có trọng số hoặc không trọng số, tùy vào loại đồ thị. Đồ thị là một cách biểu diễn mô hình hoá quan hệ giữa các đối tượng hoặc sự kiện trong thực tế bằng các đỉnh và cạnh.

Một số khái niệm cơ bản trong đồ thị bao gồm:

1. Đỉnh (Vertex): Là các điểm trong đồ thị, thường được biểu diễn bằng các ký tự, số hoặc biểu tượng.
2. Cạnh (Edge): Là các đường nối giữa các đỉnh trong đồ thị, có thể có hướng (được gọi là cạnh có hướng) hoặc không hướng (được gọi là cạnh vô hướng).
3. Độ dài (Weight): Là một giá trị gán cho mỗi cạnh trong đồ thị, thường biểu thị chi phí, khoảng cách, thời gian, hoặc các thuộc tính khác của mối quan hệ giữa hai đỉnh.
4. Đỉnh kề (Adjacent vertex): Hai đỉnh được nối với nhau bởi một cạnh.
5. Đường đi (Path): Là một chuỗi các cạnh nối liên tiếp giữa các đỉnh trong đồ thị.
6. Chu trình (Cycle): Là một đường đi đóng, bắt đầu và kết thúc tại cùng một đỉnh.
7. Đồ thị con (Subgraph): Là một đồ thị được tạo thành từ một tập con của đỉnh và cạnh của đồ thị gốc.
8. Cây (Tree): Là một đồ thị vô hướng, liên thông, không có chu trình.
9. Đồ thị liên thông (Connected graph): Là đồ thị mà giữa hai đỉnh bất kỳ trong đồ thị có ít nhất một đường đi.

Hàm mục tiêu (objective function) là một khái niệm trong lĩnh vực tối ưu hóa, được sử dụng để mô tả mục tiêu cần đạt được trong bài toán tối ưu. Nó là một hàm số định nghĩa trên tập hợp các biến quyết định của bài toán và được sử dụng để đánh giá mức độ tốt/xấu của các giải pháp khác nhau. Giá trị tối ưu của hàm mục tiêu thường là giá trị lớn nhất hoặc nhỏ nhất của hàm số, tùy vào yêu cầu của bài toán. Các biến quyết định trong bài toán sẽ được điều chỉnh để đạt được giá trị tối ưu của hàm mục tiêu.

1. Bài toán Người du lịch  
   2.1. Định nghĩa bài toán

Bài toán người du lịch - là một bài toán tối ưu hóa quan trọng trong lĩnh vực toán học, được sử dụng trong nhiều ứng dụng thực tế, đặc biệt là trong lĩnh vực logistics và giao thông vận tải.

Bài toán người du lịch xuất phát từ năm 1930, khi nhà toán học người Ireland - Sir William Hamilton đưa ra câu hỏi về việc có thể đi qua tất cả các thành phố của Đảo Anh một lần, và trở về thành phố xuất phát, trong khi đi qua mỗi thành phố chỉ một lần duy nhất. Tuy nhiên, bài toán người du lịch không được chú ý đến cho đến năm 1959, khi nhà toán học người Mỹ Merrill Flood và Melvin Dresher đưa ra một phiên bản của bài toán này.

2.2. Ứng dụng thực tế của bài toán Người du lịch

bài toán người du lịch là một trong những bài toán tối ưu hóa quan trọng nhất trong lĩnh vực toán học, được nghiên cứu và ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau, bao gồm kế hoạch sản xuất, lập lịch sản xuất, kế hoạch vận chuyển, và quản lý cơ sở hạ tầng.

VD :

* Quản lý chuỗi cung ứng (tối ưu hóa đường vận chuyển).
* Lập lịch trình cho robot trong nhà máy sản xuất.
* Thiết kế mạng lưới giao thông đô thị.

A person and person looking at a diagram

Description automatically generated with medium confidence

Bài toán người du lịch

Mặc dù bài toán người du lịch được đưa ra từ rất lâu, nhưng đến nay vẫn chưa có giải pháp tối ưu cho bài toán này, và nó vẫn là một thách thức lớn đối với các nhà toán học và nhà khoa học máy tính trên toàn thế giới. Tuy nhiên, nhiều phương pháp được đề xuất để giải quyết bài toán người du lịch, bao gồm các thuật toán tìm kiếm cục bộ, thuật toán di truyền, và thuật toán quy hoạch tuyến tính.  
3.3. Thuật toán di truyền (Genertic Algorithm)

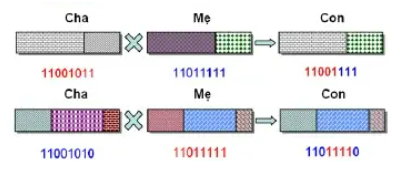
Giải thuật di truyền (GA-Genetic Algorithm) là kỹ thuật phỏng theo quá trình thích nghi tiến hóa của các quần thể sinh học dựa trên học thuyết Darwin. GA là phương pháp tìm kiếm tối ưu ngẫu nhiên bằng cách mô phỏng theo sự tiến hóa của con người hay của sinh vật. Tư tưởng của thuật toán di truyền là mô phỏng các hiện tượng tự nhiên, là kế thừa và đấu tranh sinh tồn.

Thuật toán di truyền gồm có 4 quy luật cơ bản là lai ghép, đột biến, sinh sản và chọn lọc tự nhiên như sau:

**3.3.1. Quá trình lai ghép (phép lai)**

Quá trình này diễn ra bằng cách ghép một hay nhiều đoạn gen từ hai nhiễm sắc thể cha-mẹ để hình thành nhiễm sắc thể mới mang đặc tính của cả cha lẫn mẹ. Phép lai này có thể mô tả như sau:

* Chọn ngẫu nhiên hai hay nhiều cá thể trong quần thể. Giả sử chuỗi nhiễm sắc thể của cha và mẹ đều có chiều dài là m.
* Tìm điểm lại bằng cách tạo ngẫu nhiên một con số từ 1 đến m-1. Như vậy, điểm lai này sẽ chia hai chuỗi nhiễm sắc thể cha-mẹ thành hai nhóm nhiễm sắc thể con là và . Hai chuỗi nhiễm sắc thể con lúc này sẽ là và
* Đưa hai chuỗi nhiễm sắc thể con vào quần thể để tiếp tục tham gia quá trình tiến hóa



Lai ghép 1 điểm cắt, nhiều điểm cắt

A picture containing screenshot, text, line, drive

Description automatically generated

Lai ghép nhiều đoạn

##### **3.3.2. Quá trình đột biến (phép đột biến)**

Quá trình tiến hóa được gọi là quá trình đột biến khi một hoặc một số tính trạng của con không được thừa hưởng từ hai chuỗi nhiễm sắc thể cha-mẹ. Phép đột biến xảy ra với xác suất thấp hơn rất nhiều lần so với xác suất xảy ra phép lai. Phép đột biến có thể mô tả như sau:

* Chọn ngẫu nhiên một số k từ khoảng
* Thay đổi giá trị của gen thứ k
* Đưa nhiễm sắc thể con vào quần thể để tham gia quá trình tiến hóa tiếp theo

##### **3.3.3. Quá trình sinh sản và chọn lọc (phép tái sinh và phép chọn)**

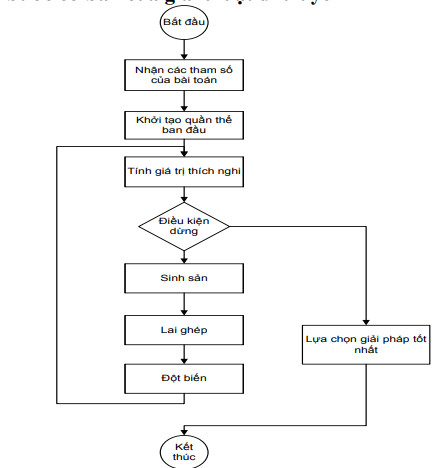
***Phép tái sinh*:** là quá trình các cá thể được sao chép dựa trên độ thích nghĩ của nó. Độ thích nghi là một hàm được gần các giá trị thực cho các cả thể trong quần thể của nó. Phép tài sinh có thể mô phỏng như sau:

* Tính độ thích nghi của từng cá thể trong quần thể, lập bảng cộng dồn các giá trị thích nghi đó (theo thứ tự gần cho từng cả thể) ta được tổng độ thích nghi. Gia sử quẩn thể có n cá thể. Gọi độ thích nghi của cả thể thứ i là , tổng dồn thứ i là . Tổng độ thích nghi là
* Tạo số ngẫu nhiên F có giá trị trong đoạn từ 0 đến
* Chọn cả thể k đầu tiên thỏa mãn F đưa vào quần thể của thế hệ mới.

***Phép chọn*:** là quá trình loại bỏ các cá thể xấu và để lại những cá thể tốt. Phép chọn được mô tả như sau:

* Sắp xếp quần thể theo thứ tự độ thích nghi giảm dần
* Loại bỏ các cá thể cuối dây, chỉ để lại n cá thể tốt nhất

***Sơ đồ cấu trúc của giải thuật di truyền:***



***Cấu trúc thuật giải di truyền tổng quát:***

Bắt đầu

t=0;

Khởi tạo P(t);

Tính độ thích nghi cho các cá thể thuộc P(t);

Khi (điều kiện dừng chưa thỏa) lặp

t = t + 1;

Chọn lọc P(t);

Lai P(t);

Đột biến P(t);

Hết lặp;

Kết thúc

**III. PHÁT BIỂU BÀI TOÁN VÀ CÀI ĐẶT GIẢI THUẬT DI TRUYỀN**

1. Phát biểu bài toán

Có n địa điểm cần được giao hàng. Người giao hàng xuất phát từ 1 địa điểm và thời gian xuất phát được chọn, đi qua các địa điểm khác để giao hàng và trở về địa điểm ban đầu. Mỗi địa điểm chỉ đến 1 lần, và thời gian di chuyển từ 1 địa điểm đến các địa điểm kế tiếp liền kề với nó được cho trước. Biết rằng trong quá trình di chuyển, người giao hàng có thể sẽ gặp một số đoạn đường bị tắc trong một khung giờ nhất định. Hãy tìm ra 1 chu trình và thời gian di chuyển (một đường đi khép kín thỏa mãn các điều kiện trên) sao cho thời gian di chuyển là ngắn nhất.

Bài toán được biểu diễn như 1 đồ thị vô hướng có trọng số, trong đó mỗi địa điểm là một đỉnh của đồ thị. Thời gian di chuyển của 2 thành phố là độ dài cạnh. Trong 1 khung giờ nhất định sẽ có những đường màu đỏ biểu hiện các đường đi có xuất hiện ách tắc. Đây là vấn đề cực tiểu hóa với điểm đầu kèm theo thời gian xuất phát và điểm cuối là cùng 1 điểm sau khi thăm hết các đỉnh còn lại đúng 1 lần.

A black background with white circles and a red line

Description automatically generated with low confidence

Ví dụ mô tả đồ thị của bài toán

Bài toán biểu diễn bằng toán học có đồ thị vô hướng G=(V,E) với V={v0,v1,…,vn-1} là tập các đỉnh và nút và E{(vi,vj): vi,vj }là tập hợp các cạnh. Mỗi đỉnh vi đại diện cho 1 địa điểm vi và mỗi cạnh (vi,vj) là khoảng cách giữa 2 thành phố i, j. Có K{(vi,vj),: vi,vj } là tập hợp các cạnh có xuất hiện ách tắc.

Một chu trình cần tìm gọi là chu trình Hamilton. Chu trình này luôn đi qua các đỉnh của đồ thị đúng 1 lần và kết thúc tại điểm bắt đầu.

Bài toán tìm cực tiểu: Min F = với i,j=n, là thời gian di chuyển từ i đến j, K*ij* là thời gian tắc đường giữa địa điểm i và địa điểm j.

1. Phân tích bài toán

Cấu trúc sơ bộ của bài toán:

* Đầu vào: Tập hợp các địa điểm giao hàng, địa điểm bắt đầu, thời gian xuất phát.
* Đầu ra: Chu trình đường đi và thời gian di chuyển qua tất cả các điểm trong thời gian ngắn nhất.
* Yêu cầu: Xét đến yếu tố tắc đường trong quá trình di chuyển.

Phân tích yêu cầu:

* Địa điểm xuất phát luôn được cho trước và kết thúc tại điểm bắt đầu.
* Thời gian xuất phát có thể là thời gian thực hoặc được nhập vào.
* Không có địa điểm nào mà không có đường đi tới địa điểm khác.
* Coi vận tốc của quá trình đi là không đổi trong mọi điều kiện di chuyển.
* Không xét đến những yếu tố ngoại cảnh khác: thời tiết, sử dụng nguyên liệu, điều phối giao thông, khối lượng hàng hóa, thời gian nhận hàng…
* Với yêu cầu ràng buộc có yếu tố tắc đường trong quá trình di chuyển, ta sẽ cho 1 số quãng đường tắc đó xuất hiện trong 1 khung giờ nhất định và thời gian so với khi không tắc đường là chậm hơn bao nhiêu.
* Mỗi địa điểm trên đường đi chỉ đi qua 1 lần duy nhất
* Lộ trình tìm được có thể không phải là tối ưu nhất mà chỉ mang tính chất tương đối có thể chấp nhận được.

1. Thiết kế và cài đặt thuật toán  
   3.1. Mã hóa bài toán và khởi tạo lớp đối tượng

##### **3.1.1. Mã hóa bài toán**

Dữ liệu bài toán được mã hóa như sau: Bài toán có thể biểu diễn đồ thị vô hướng có trọng số M = [mij] với (i,j = ). Ví dụ:

3.2 **Khởi tạo các biến và kiểu dữ liệu**

Thuật toán di truyền sử dụng các biến với kiểu kiểu dữ liệu đi kèm được thể hiện ở bảng dưới đây:

|  |  |  |
| --- | --- | --- |
| **Tên biến** | **Kiểu dữ liệu** | **Ý nghĩa** |
| individual | Int | Số lượng cá thể |
| result | ArrayList<Integer> | 1 chu trình kết quả |
| results | ArrayList<ArrayList<Integer>> | Các chu trình kết quả |
| ran | Random | Khởi tạo số ngẫu nhiên |
| Fitness | ArrayList<LocalTime> | Độ tương thích của các chu trình |
| countGeneration | Int | Số lượng đời |

3.**3.** **Mã hóa chu trình (cá thể - gen)**

Chu trình được mã hóa bằng mảng có thứ tự các số hiệu của đỉnh. Với đồ thị n đỉnh thì mảng có kích thước n+1 phần tử. Ví dụ với chu trình 7 đỉnh:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 2 | 5 | 4 | 6 | 7 | 1 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 4 | 3 | 2 | 5 | 7 | 6 | 1 |

Ngoài ra mỗi chu trình cần có thêm thông số về chi phí của toàn bộ chu trình đó. Chi phí được tính theo phương thức EvulationFitness() (phương thức sẽ được đề cập ở bên dưới).

Mỗi chu trình là 1 lời giải, trong giải thuật di truyền coi đói như 1 cá thể. Việc tiến hóa về sau ta sẽ dựa trên tập chu trình khởi tạo ban đầu và tìm ra kết quả tốt nhất sau 1 thế hệ.

3.4. **Khởi tạo quần thể**

Để thuật toán được tính toán 1 cách nhanh chóng, quần thể ban đầu sẽ được khởi tạo bằng cách gán giá trị đầu tiên và cuối cùng của chu trình cho điểm xuất phát. Ở giữa sẽ là các chỉ số ngẫu nhiên không trùng lặp từ . Số kích thước cá thể tối đa có thể tùy biến theo số đỉnh của đồ thị cần giải, ở đây nhóm chọn kích thước là 100 cá thể.

**Cài đặt code:**

A screenshot of a computer code

Description automatically generated

3.5. **Đánh giá**

Ở phương thức này sẽ xóa dữ liệu tính toán được hiện có sau mỗi đời. Sau đó tính toán chi phí (thời gian di chuyển được) của chu trình đang xét lấy từ dữ liệu từ ma trận trọng số weightMatrix và đối tượng TrafficJam. Kết quả tính được sẽ được thêm vào results để lưu trữ. Việc tính toán và lưu trữ sẽ được thực hiện đến khi nào tính đủ số cá thể individual.

Việc tính toán thời gian diễn ra như sau:

* **Bước 1:** Kiểm tra xem có đường đi giữa đỉnh hiện tại và đỉnh tiếp theo không
* **Bước 2:** Nếu có, kiểm tra xem đoạn đường đó có nằm trong diện tắc đường hay không
* **Bước 3:** Nếu có, kiểm tra xem thời gian đi từ điểm đầu đã nằm trong khoảng thời gian tắc hay chưa
* **Bước 4:** Nếu có, cộng thời gian di chuyển bằng thời gian tắc đường
* **Bước 5:** Kiểm tra xem thời gian đi từ điểm đầu đã vượt qua thời gian tắc đường hay chưa
* **Bước 6:** Nếu có, cộng thời gian di chuyển bằng thời gian chậm đi trong đoạn đường tắc đường
* **Bước 7:** Sau khi hoàn thành kiểm tra cho đoạn đường đó thì cộng thời gian di chuyển bằng thời gian di chuyển giữa hai đỉnh
* **Bước 8:** Nếu không có đường đi giữa hai đỉnh, cộng thời gian di chuyển bằng 1 giờ

**Cài đặt code:**

|  |
| --- |
| public static void EvulationFitness() {  Fitness.clear();  for (int i = 0; i < individual; i++) {  Fitness.add(i, TSPProblem.getStartTime());  ArrayList<Integer> result = results.get(i);  int k;  for (k = 0; k < (TSPProblem.getN()); k++) {  if ((TSPProblem.getWeightMatrix()[result.get(k)]  [result.get(k + 1)] > 0)) {  for (TrafficJam trafficJam:TSPProblem  .getTrafficJams()) {  If (trafficJam.getSegment().equals(TSPProblem.getVertexNames()[result.get(k)] + TSPProblem.getVertexNames()[result.get(k + 1)])) {  If (Fitness.get(i).isAfter(trafficJam  .getJamStartTime()) && Fitness.get(i).isBefore(trafficJam.getJamEndTime())) {  Fitness.set(i, Fitness.get(i).plusMinutes(TrafficJam.getDelay()));  }  if (Fitness.get(i).isBefore(trafficJam.getJamStartTime())) {  if (Fitness.get(i)  .plusMinutes(TSPProblem.getWeightMatrix()[result.get(k)][result.get(k + 1)])  .isAfter(trafficJam.getJamStartTime())) {  Fitness.set(i, Fitness.get(i).plusMinutes(TrafficJam.getDelay()));  }  }  }  }  Fitness.set(i, Fitness.get(i).plusMinutes(TSPProblem.getWeightMatrix()[result.get(k)][result.get(k + 1)]));  } else {  Fitness.set(i, Fitness.get(i).plusHours(1));  }  }  }  } |

3.6. **Chọn lọc tự nhiên**

Cách thức chọn lọc cá thể được đánh giá dựa trên chi phí của mỗi chu trình. Cá thể được chọn làm lời giải cuối cùng là cá thể có chi phí nhỏ nhất trong quần thể.

Nhóm đưa ra cách chọn lọc tự nhiên như sau:

* Sắp xếp đánh giá quần thể Fitness theo chi phí tăng dần
* Lấy ngưỡng 80% của Fitness
* Loại bỏ những cá thể có độ thích nghi lớn hơn ngưỡng
* Việc loại bỏ sẽ kết thúc khi không còn cá thể nào lớn hơn ngưỡng

Với cách thức chọn lọc này sẽ loại bỏ nhanh những cá thể có độ thích nghi thấp nhưng đồng thời sẽ làm mất sự đa dạng nguồn gen cho tiến hóa cho các thế hệ sau so với việc tiến hóa rồi mới chọn lọc tự nhiên.

**Cài đặt code:**

|  |
| --- |
| public static void Selection() {  @SuppressWarnings("unchecked")  ArrayList<LocalTime> temp = (ArrayList<LocalTime>) Fitness.clone();  Collections.sort(temp);  LocalTime mark = temp.get(individual \* 80 / 100);  int check = individual;  for (int i = 0; i < check; i++) {  while (Fitness.get(i).isAfter(mark)) {  results.remove(i);  Fitness.remove(i);  if (i >= results.size()) {  i--;  }  check--;  }  }  } |

**3.7. Lai ghép**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 2 | 5 | 4 | 6 | 7 | 1 |

Phương thức lai ghép sẽ được thực hiện bằng cách lấy ngẫu nhiên 2 cá thể:

Cha:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 7 | 6 | 2 | 5 | 4 | 3 | 1 |

Mẹ:

Thực hiện phép lai ghép 1 điểm cắt với vị trí cắt là ngẫu nhiên trừ 2 điểm đầu và cuối:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 4 | 6 | 7 |  |  |  |  |

* Cắt từ điểm cut đến điểm trước điểm cuối của chu trình cha đưa vào chu trình con đã có sẵn điểm đầu là điểm xuất phát, lấy 1 ví dụ cut = 4:

Con:

* Xét từ các điểm lần lượt trừ điểm đầu và cuối của chu trình mẹ, nạp dần các điểm chưa có trong con lai theo thứ tự duyệt và chèn điểm xuất phát vào cuối, ta được:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 4 | 6 | 7 | 2 | 5 | 3 | 1 |

\* Cách lai ghép này đảm bảo con lai mới là 1 chu trình thỏa mãn

Việc lai ghép sẽ được thực hiện liên tục cho đến khi kích thước quần thể được trở về ban đầu

**Cài đặt code:**

|  |
| --- |
| public static void Hybridize() {  while (results.size() != individual) {  ArrayList<Integer> dad = (ArrayList<Integer>) results.get(ran.nextInt(results.size())).clone();  ArrayList<Integer> mom = (ArrayList<Integer>) results.get(ran.nextInt(results.size())).clone();  int cut = ran.nextInt((TSPProblem.getN() - 1)) + 1;  result.clear();  result.add(dad.get(0));  for (int i = cut; i < TSPProblem.getN(); i++) {  result.add(dad.get(i));  }  for (int j = 1; j < TSPProblem.getN(); j++) {  if (result.indexOf(mom.get(j)) == -1) {  result.add(mom.get(j));  }  result.add(dad.get(0));  results.add(new ArrayList<>(result));  }  } |

3.8. Đột biến

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 2 | 5 | 4 | 6 | 7 | 1 |

Phương thức đột biến sẽ lấy 1 cá thể ngẫu nhiên để thực hiện đột biến:

C1:

Thực hiện đột biến bằng cách tráo đổi các điểm trên gen cho nhau. Số lần tráo đổi được sinh ngẫu nhiên từ 1 đến n-1, vị trí điểm tráo cũng được sinh ngẫu nhiên trừ 2 điểm đầu cuối trong quá trình chạy.

Ví dụ với đột biến C1 bằng tráo đổi 2 lần: tráo 3 và 7, tráo 2 và 5. Khi đó ta được chu trình mới:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 7 | 5 | 2 | 4 | 6 | 3 | 1 |

\* Cách đột biến này đảm bảo con lai mới là 1 chu trình thỏa mãn

**Cài đặt code:**

|  |
| --- |
| public static void Mutate() {  int pos = ran.nextInt(results.size());  result = results.get(pos);  int count = ran.nextInt((TSPProblem.getN() - 1)) + 1;  while (count > 0) {  int p1 = ran.nextInt((TSPProblem.getN() - 1)) + 1;  int p2 = ran.nextInt((TSPProblem.getN() - 1)) + 1;  do {  p2 = ran.nextInt((TSPProblem.getN() - 1)) + 1;  } while (p1 == p2);  int temp = result.get(p1);  result.set(p1, result.get(p2));  result.set(p2, temp);  count--;  }  results.set(pos, (result));  } |

##### **3.9. Phương thức in ra kết quả**

* Sắp xếp đánh giá quần thể theo chi phí tăng dần
* Lấy kết quả đầu tiên trong đánh giá quần thể
* Xét đến đời thứ countGeneration-1 của quần thể
* Xét các cá thể trong quần thể nếu cá thể nào bằng với độ thích nghi trên thì in ra
* Nhóm còn làm thêm phương thức totalTime() để thuận tiện đổi ra tổng thời gian di chuyển của chu trình từ thời điểm xuất phát

**Cài đặt code:**

|  |
| --- |
| static int totalTime(LocalTime best) {  return best.getHour() \* 60 + best.getMinute() - TSPProblem.getStartTime().getHour() \* 60 - TSPProblem.getStartTime().getMinute();  }  public static void Print(int t) {  ArrayList<LocalTime> temp = (ArrayList<LocalTime>) Fitness.clone();  Collections.sort(temp);  LocalTime best = temp.get(0);  if (t == countGeneration - 1) {  System.out.println("Final way:");  for (int i = 0; i < individual; i++) {  if (Fitness.get(i) == best) {  for (int j = 0; j <= TSPProblem.getN(); j++) {  System.out.print(TSPProblem.NameList.get(results.get(i).get(j)));  if (j != TSPProblem.getN()) {  System.out.printf(" -> ");  }  }  }  }  System.out.println();  System.out.println("Start time: " + TSPProblem.getStartTime());  System.out.println("End time: " + best);  System.out.println("Total time watse: " + totalTime(best));  }  } |

**3.10. Phương thức tổng hợp và chạy chương trình**

Phương thức sẽ đọc file dữ liệu rồi sẽ khởi tạo đối tượng TSPProblem từ file.

Phương thức sẽ khởi tạo quần thể ban đầu rồi sẽ cho diễn ra trong countGeneration đời, càng lâu đời thì kết quả càng tối ưu. Ở đây nhóm sẽ để là 100 đời.

**Cài đặt code:**

|  |
| --- |
| public static void main(String[] args) throws IOException {  String file = "src\\data\\data.txt";  TSPProblem.readTSPProblemFromFile(file);  InitialPopulation();  for (int j = 0; j < countGeneration; j++) {  EvulationFitness();//Khoi tao  Print(j);  Selection();//Chon loc  Hybridize();//Lai  Mutate();//Dot bien  }  } |

4. kiểm thử

|  |  |
| --- | --- |
| Input | Genetic\_Algorithm |
| A picture containing text, screenshot, font, number  Description automatically generated | A screenshot of a computer program  Description automatically generated with low confidence |
| A picture containing text, screenshot, font, number  Description automatically generated | A screenshot of a computer program  Description automatically generated with low confidence |
| A picture containing text, screenshot, font, number  Description automatically generated | A screenshot of a computer  Description automatically generated with low confidence |
| A picture containing text, screenshot, font, number  Description automatically generated | A screenshot of a computer  Description automatically generated with low confidence |
| A picture containing text, screenshot, font, number  Description automatically generated | A picture containing text, screenshot, font  Description automatically generated |

**IV. KẾT LUẬN**

1. Kết quả đạt được

Báo cáo áp dụng thuật toán di truyền để giải quyết bài toán người giao hàng được thêm vào 1 số điều kiện đặc biệt để tìm ra chu trình ngắn nhất để đi lại giữa các địa điểm.

1. Hạn chế và hướng phát triển  
   2.1. Hạn chế

* Thuật toán di truyền vẫn còn dựa vào độ “may mắn” để ra kết quả tối ưu
* Đối với cách tiếp cận của thuật toán di truyền thì kết quả đưa ra có thể chưa được tối ưu nhất so với thuật toán nhánh cận mà chỉ là cận tối ưu
* Thuật toán di truyền có thể tính toán không tốt với thời gian đi nhiều và thời gian xuất phát muộn

2.2. Hướng phát triển

* Tối ưu thuật toán có thể tìm ra kết quả chính xác 1 cách nhanh chóng hơn
* Phát triển thuật toán di truyền có thể tính toán trong thời gian lâu hơn
* Có thể làm chương trình mô phỏng googlemap với dữ liệu trên

**TÀI LIỆU THAM KHẢO**