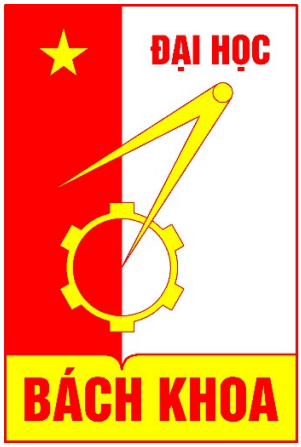
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

_____ * _____



TRÍ TUỆ NHÂN TẠO

Giải thuật di truyền với bài toán người du lịch

Sinh viên thực hiện	**Trương Công Phú**	**20101991**
	Cấn Kim Tùng	**20102465**
	Nguyễn Hoàng Long	**20101802**
	Đinh Quang Vinh	**20102786**
	Trần Hữu Sơn	**20102109**
	Đào Trọng Huấn	**20101600**

Hà Nội - 2013

MỤC LỤC

LỜI NÓI ĐẦU

Bài toán người du lịch là một trong những bài toán được nghiên cứu sâu nhất trong lĩnh vực tối ưu hóa. Báo cáo này sẽ trình bày 1 hướng tiếp cận giải quyết bài toán người du lịch sử dụng giải thuật di truyền.

Giải thuật di truyền về cơ bản muốn mô phỏng lại quá trình tiến hóa của sinh vật trong tự nhiên vào các bài toán tối ưu hóa từ đó đưa ra lời giải tốt (có thể không là tối ưu nhất) khi mà không thể đưa ra được 1 giải thuật chính xác hay việc vét cạn các trường hợp là bất khả thi.

Mặc dù đã rất cố gắng nhưng vẫn không thể tránh khỏi những sai sót, mong thầy giáo chỉ bảo thêm.

CHƯƠNG I : GIẢI THUẬT DI TRUYỀN (Genetic Algorithm - GA)

Giải thuật di truyền cũng như tiến hóa dựa trên khái niệm cho rằng quá trình tiến hóa tự nhiên là hoàn hảo nhất, hợp lý nhất và tự nó đã mang tính tối ưu. Sự tối ưu đó được thể hiện ở chỗ thế hệ sau bao giờ cũng phát triển tốt hơn thế hệ trước. Tiến hóa tự nhiên được duy trì nhờ hai quá trình cơ bản: sinh sản và chọn lọc tự nhiên, xuyên suốt quá trình tiến hóa tự nhiên, các thế hệ mới luôn được sinh ra để bổ sung thay thế thế hệ cũ, cá thể nào thích ứng với môi trường sẽ tồn tại, ngược lại sẽ bị đào thải.

Giải thuật di truyền bao gồm 4 bước chính: Mã hóa lời giải, khởi tạo quần thể, sử dụng các phép toán di truyền và đánh giá độ thích nghi. Sau đó, chúng ta lại sinh ra một quần thể mới bằng phép chọn lọc rồi tiếp tục sử dụng các phép toán di truyền và đánh giá độ thích nghi của các cá thể (điển hình bởi nhiễm sắc thể - NST) trong quần thể. Thuật giải được thực hiện qua càng nhiều thế hệ thì lời giải đưa ra càng tối ưu.

1. Động lực

Thuật giải di truyền cung cấp một phương pháp học được thúc đẩy bởi sự tương tự với sự tiến hóa sinh học. Thay vì tìm kiếm các giả thuyết từ tổng quát đến cụ thể hoặc từ đơn giản đến phức tạp, GAs tạo ra các giả thuyết kế tiếp bằng cách lặp việc đột biến và việc tái hợp các phần của giả thuyết được biết hiện tại là tốt nhất. Ở mỗi bước, một tập các giả thuyết được gọi là quần thể hiện tại được cập nhật bằng cách thay thế vài phần nhỏ quần thể bởi cá thể con của các giả thuyết tốt nhất ở thời điểm hiện tại. Sự phổ biến của GAs được thúc đẩy bởi các yếu tố sau:

- Tiến hóa là một phương pháp mạnh, thành công cho sự thích nghi bên trong các hệ thống sinh học.

- GA có thể tìm kiếm trên các không gian giả thuyết có các phần tương tác phức tạp, ở đó ảnh hưởng của mỗi phần lên toàn thể độ thích nghi giả thuyết khó có thể mô hình.
- Thuật giải GA có thể được thực hiện song song và có thể tận dụng thành tựu của phần cứng máy tính mạnh.

2. Thuật giải di truyền

Bài toán dành cho GAs là tìm kiếm trên không gian các giả thuyết ứng cử để xác định giả thuyết tốt nhất. Trong GAs "giả thuyết tốt nhất" được định nghĩa như là một giả thuyết tối ưu hóa một đại lượng số được định nghĩa trước cho bài toán sắp tới, được gọi là độ thích nghi của giả thuyết. Ví dụ, nếu tác vụ học hỏi là bài toán xấp xỉ một hàm chưa biết cho tập mẫu huấn luyện gồm dữ liệu đầu vào và dữ liệu đầu ra, thì độ thích nghi có thể được định nghĩa như là độ chính xác của giả thuyết trên dữ liệu huấn luyện này. Nếu tác vụ là học chiến lược chơi cờ, độ thích nghi có thể là số ván thắng của chiến lược này khi đấu với các chiến lược khác trong quần thể hiện tại.

Mặc dù các thuật giải di truyền được thực hiện thay đổi theo bài toán cụ thể, nhưng chúng chia sẻ chung cấu trúc tiêu biểu sau: Thuật giải hoạt động bằng cách cập nhật liên tục tập giả thuyết – được gọi là quần thể. Ở mỗi lần lặp, tất cả các cá thể trong quần thể được ước lượng tương ứng với hàm thích nghi. Rồi quần thể mới được tạo ra bằng cách lựa chọn có xác suất các cá thể thích nghi tốt nhất từ quần thể hiện tại. Một số trong những cá thể được chọn được đưa nguyên vẹn vào quần thể kế tiếp. Những cá thể khác được dùng làm cơ sở để tạo ra các cá thể con bằng cách áp dụng các tác động di truyền: lai ghép và đột biến.

GA(**Fitness**, **Fitness_threshold**, **p**, **r**, **m**)

{

// **Fitness**: *hàm gán thang điểm ước lượng cho một giả thuyết*

// **Fitness_threshold**: *Ngưỡng xác định tiêu chuẩn dừng giải thuật tìm kiếm*

// **p**: *Số cá thể trong quần thể giả thuyết*

// **r**: *Phân số* cá thể trong quần thể được áp dụng toán tử lai ghép ở mỗi bước

// **m**: *Tỉ lệ cá thể bị đột biến*

- Khởi tạo quần thể: **P** ← Tạo ngẫu nhiên **p** cá thể giả thuyết
- Ước lượng: Ứng với mỗi **h** trong **P**, tính *Fitness*(**h**)
- **while** [**max** *Fitness*(**h**)] < *Fitness_threshold* **do**

  Tạo thế hệ mới, $\mathbf{P_S}$

  1. ***Chọn cá thể***: chọn theo xác suất **(1 – r)p** cá thể trong quần thể **P** thêm vào $\mathbf{P_S}$. Xác suất $\mathbf{Pr(h_i)}$ của giả thuyết $\mathbf{h_i}$ thuộc **P** được tính bởi công thức:

     $$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

  2. ***Lai ghép***: chọn lọc theo xác suất $\frac{r \times p}{2}$ cặp giả thuyết từ quần thể **P**, theo $\mathbf{Pr(h_i)}$ đã tính ở bước trên. Ứng với mỗi cặp $\mathbf{<h_1, h_2>}$, tạo ra hai con bằng cách áp dụng toán tử lai ghép. Thêm tất các các con vào $\mathbf{P_S}$.
  3. ***Đột biến***: Chọn **m%** cá thể của $\mathbf{P_S}$ với xác suất cho mỗi cá thể là như nhau. Ứng với mỗi cá thể biến đổi một bit được chọn ngẫu nhiên trong cách thể hiện của nó.
  4. ***Cãp nhật***: **P** ← $\mathbf{P_S}$.
  5. ***Ước lượng***: Ứng với mỗi h trong P, tính *Fitness*(h)

- Trả về giả thuyết trong P có độ thích nghi cao nhất.

}

Figure 1 : Các bước cơ bản của giải thuật

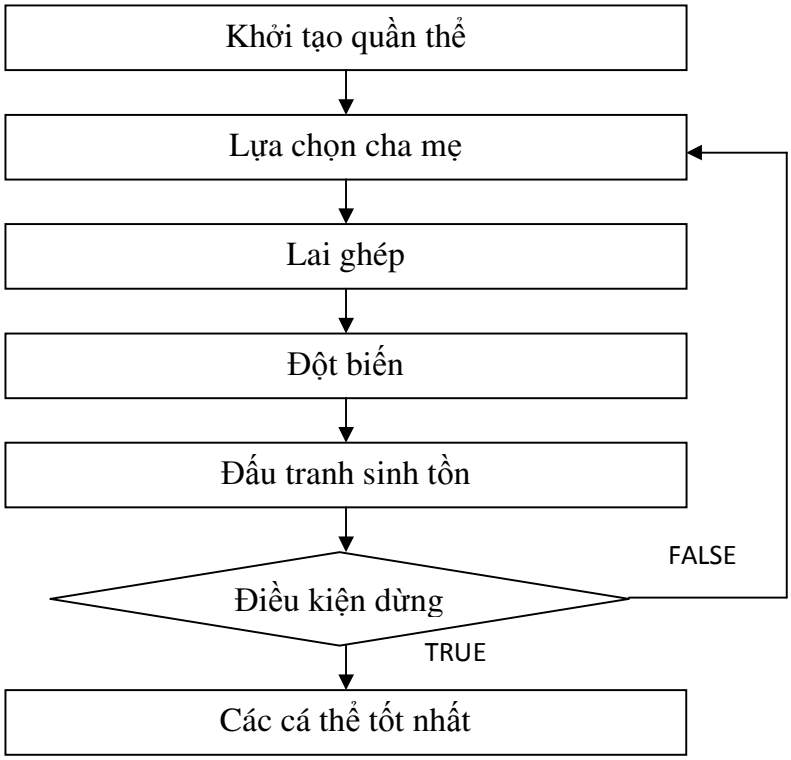


Figure 2 : Lưu đồ giải thuật cơ bản

3. Các toán tử di truyền

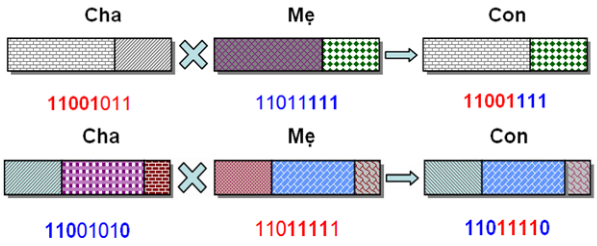
3.1 Lai ghép :

Phép lai là quá trình hình thành NST mới trên cơ sở NST cha mẹ, bằng cách ghép một hay nhiều đoạn gen của hai (hay nhiều) NST cha mẹ khác nhau.

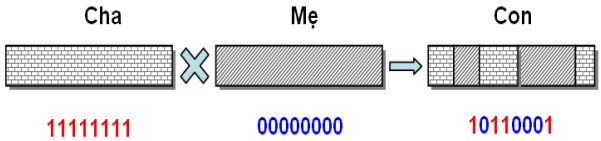
Các cặp cha mẹ được lựa chọn ngẫu nhiên và xác suất xảy ra lai ghép với mỗi cặp được quy định từ trước.

Có nhiều cách lai ghép khác nhau:

- Lai ghép một điểm cắt, nhiều điểm cắt.



- Lai ghép nhiều đoạn.



3.2 Đột biến :

Đột biến là tình trạng NST con không có một (hoặc một số) tính trạng có trong mã di truyền của cha mẹ.

Các cặp cha mẹ được lựa chọn ngẫu nhiên và xác suất xảy ra đột biến với mỗi cặp được quy định từ trước, thường là rất nhỏ.

Các phép đột biến thường được sử dụng:

- Đảo bit.
- Hoán vị: Đổi vị trí của các gen với nhau.
- Đổi giá trị: Thay đổi giá trị tại một điểm gen.
- Đảo đoạn: Đảo thứ tự của một đoạn NST bất kì.

4. Đấu tranh sinh tồn

Chọn những NST từ quần thể kết quả theo một quy tắc nào đó thay thế cho cha mẹ để sinh ra thế hệ mới. Một số phương thức đấu tranh sinh tồn cơ bản:

- Tráo đổi hoàn toàn cha mẹ bằng con.
- Tráo đổi ngẫu nhiên: Chọn ngẫu nhiên k cha mẹ và thay thế bằng k con mới.
- Chọn những cá thể ưu tú nhất trong quần thể.

CHƯƠNG II : BÀI TOÁN NGƯỜI DU LỊCH (Travelling Salesman Problem - TSP)

1. Lịch sử bài toán :

Bài toán người bán hàng (tiếng Anh: travelling salesman problem - TSP) là một bài toán NP-Hard thuộc thể loại tối ưu tổ hợp được nghiên cứu trong lý thuyết khoa học máy tính. Nội dung bài toán có thể hiểu khái quát như sau : Cho trước một danh sách các thành phố và khoảng cách giữa chúng, tìm chu trình ngắn nhất đi qua tất cả các thành phố đúng 1 lần.

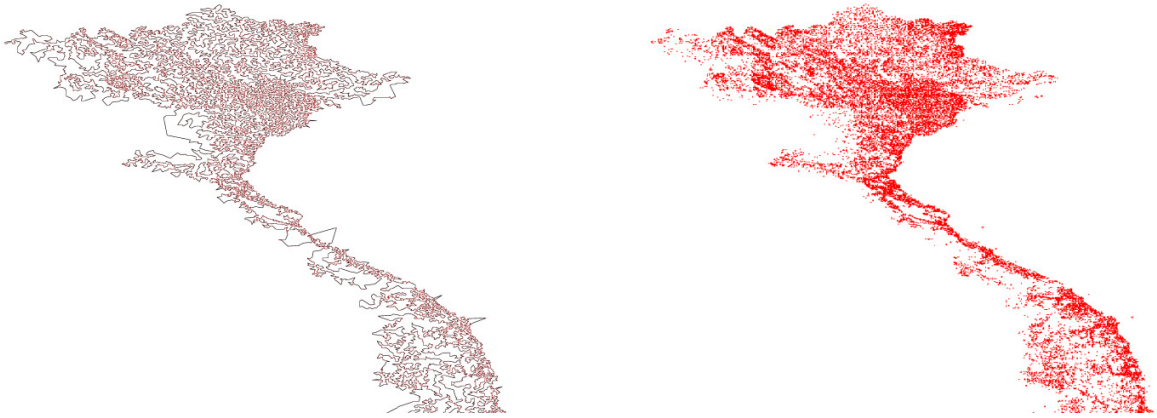


Figure 3 22,775 Cities in Vietnam, derived from data from the National Imagery and Mapping Agency database of geographic feature names.

Bài toán được nêu ra lần đầu tiên năm 1930 và là một trong những bài toán được nghiên cứu sâu nhất trong tối ưu hóa. Nó thường được dùng làm thước đo cho nhiều phương pháp tối ưu hóa. Mặc dù bài toán rất khó giải trong trường hợp tổng quát, có nhiều phương pháp giải chính xác cũng như heuristic đã được tìm ra để giải quyết một số trường hợp có tới hàng chục nghìn thành phố.

Ngay trong hình thức phát biểu đơn giản nhất, bài toán TSP đã có nhiều ứng dụng trong lập kế hoạch, hậu cần, cũng như thiết kế vi mạch, …

Nguồn gốc của bài toán người bán hàng vẫn chưa được biết rõ. Một cuốn sổ tay dành cho người bán hàng xuất bản năm 1832 có đề cập đến bài toán này và

có ví dụ cho chu trình trong nước Đức và Thụy Sĩ, nhưng không chứa bất kì nội dung toán học nào.

Bài toán người bán hàng được định nghĩa trong thế kỉ 19 bởi nhà toán học Ireland William Rowan Hamilton và nhà toán học Anh Thomas Kirkman. Trường hợp tổng quát của TSP có thể được nghiên cứu lần đầu tiên bởi các nhà toán học ở Vienna và Harvard trong những năm 1930.

Hassler Whitney ở đại học Princeton đưa ra tên bài toán người bán hàng ngay sau đó.

Trong những năm 1950 và 1960, bài toán trở nên phổ biến trong giới nghiên cứu khoa học ở Châu Âu và Mỹ. George Dantzig, Delbert Ray Fulkerson và Selmer M. Johnson ở công ty RAND tại Santa Monica đã có đóng góp quan trọng cho bài toán này, biểu diễn bài toán dưới dạng quy hoạch nguyên và đưa ra phương pháp mặt phẳng cắt để tìm ra lời giải. Với phương pháp mới này, họ đã giải được tối ưu một trường hợp có 49 thành phố bằng cách xây dựng một chu trình và chứng minh rằng không có chu trình nào ngắn hơn. Trong những thập niên tiếp theo, bài toán được nghiên cứu bởi nhiều nhà nghiên cứu trong các lĩnh vực toán học, khoa học máy tính, hóa học, vật lý, và các ngành khác.

Năm 1972, Richard M. Karp chứng minh rằng bài toán chu trình Hamilton là NP-đầy đủ, kéo theo bài toán TSP cũng là NP-đầy đủ. Đây là một lý giải toán học cho sự khó khăn trong việc tìm kiếm chu trình ngắn nhất.

Một bước tiến lớn được thực hiện cuối thập niên 1970 và 1980 khi Grötschel, Padberg, Rinaldi và cộng sự đã giải được những trường hợp lên tới 2392 thành phố, sử dụng phương pháp mặt phẳng cắt và nhánh cận.

Trong thập niên 1990, Applegate, Bixby, Chvátal, và Cook phát triển một chương trình mang tên Concorde giải được nhiều trường hợp có độ lớn kỉ lục hiện nay. Gerhard Reinelt xuất bản một bộ dữ liệu các trường hợp có độ khó khác nhau mang tên TSPLIB năm 1991, và nó đã được sử dụng bởi nhiều nhóm

nghiên cứu để so sánh kết quả. Năm 2005, Cook và cộng sự đã giải được một trường hợp có 33810 thành phố, xuất phát từ một bài toán thiết kế vi mạch. Đây là trường hợp lớn nhất đã được giải trong TSPLIB.

Đến nay bài toán TSP vẫn được tiếp tục nghiên cứu tìm ra lời giải cho các bộ dữ liệu lớn hơn. Chẳng hạn bộ dữ liệu của nước Mĩ với 115,475 thành phố người giải ra chu trình tối ưu được trao thưởng 500$ (thông tin chi tiết tại http://www.tsp.gatech.edu/).

2. Phát biểu bài toán :

(*) Các khái niệm cơ bản về đồ thị sẽ không được trình bày trong báo cáo.

Phát biểu bài toán : Cho đồ thị đầy đủ n đỉnh vô hướng, có trọng số G = (V, E). Tìm chu trình $v_1 \to v_2 \to \dots v_n \to v_1$ với $v_i \in V, i = \overline{1,n}$ sao cho tổng trọng số hành trình trên các cạnh (v_i, v_{i+1}) và (v_n, v_1) là nhỏ nhất.

Một chu trình như vậy còn gọi là chu trình Hamilton, nó đi qua tất cả các đỉnh của đồ thị đúng 1 lần. Đồ thị đầy đủ, luôn tồn tại chu trình Hamilton.

3. Phân tích độ phức tạp :

Bài toán TSP thuộc lớp bài toán NP-Khó (lớp các bài toán không có giải thuật trong thời gian đa thức).

Việc thực hiện liệt kê hết tất cả các chu trình là điều gần như không thể với số đỉnh lớn (đồ thị n đỉnh phải duyệt n! chu trình). Số chu trình phải duyệt tăng rất nhanh khi số đỉnh n càng lớn. Ngay với 1 đồ thị 100 đỉnh, việc duyệt toàn bộ cũng là điều rất khó thực hiện.

CHƯƠNG III : ĐỀ XUẤT GIẢI THUẬT DI TRUYỀN GIẢI BÀI TOÁN NGƯỜI DU LỊCH

1. Giải thuật đề xuất :

Nhóm đề xuất giải thuật di truyền đơn giản giải bài toán người du lịch. Giải thuật được cài đặt bằng ngôn ngữ java.

Các bộ dữ liệu kiểm thử được lấy tại http://www.tsp.gatech.edu/ (cung cấp các bộ dữ liệu chuẩn trên thực tế)

1.1 Mã hóa bài toán :

1.1.1 Mã hóa đồ thị :

Đồ thị được mã hóa bằng danh sách mảng các điểm và tọa độ tương ứng của chúng. Dưới đây là ví dụ về bộ dữ liệu đồ thị chuẩn.

NAME : xit1083 COMMENT : Bonn VLSI data set with 1083 points COMMENT : Uni Bonn, Research Institute for Discrete Math COMMENT : Contributed by Andre Rohe TYPE : TSP DIMENSION : 1083 EDGE_WEIGHT_TYPE : EUC_2D NODE_COORD_SECTION		
1	0	105
2	0	111
3	0	15
…	…	…

Trọng số trong cột đầu tiên là số hiệu của đỉnh, trong số thứ 2 là hoành độ, trọng số thứ 3 là tung độ. Khoảng cách giữa 2 đỉnh $M(x_i, y_i)$ và $N(x_j, y_j)$ của đồ thị (trọng số cho cạnh) được tính theo công thức :

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

1.1.2 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 phần tử. Ví dụ chu trình của đồ thị 10 đỉnh :

C1	1	3	2	4	6	9	8	7	5	10

C2	10	7	9	6	4	2	8	3	1	5

Ngoài ra mỗi chu trình cần phải có thêm thông số về chi phí của toàn bộ chu trình đó. Chi phí này được tính bằng tổng độ dài tất cả các cạnh tạo nên chu trình đó (theo công thức tính khoảng cách đã đề cập ở trên).

Mỗi chu trình là 1 lời giải, trong giải thuật di truyền coi đó 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 một số thế hệ.

Dưới đây là ví dụ về bộ dữ liệu chu trình chuẩn.

```
NAME: xit1083
COMMENT: Tour length 3558
TYPE: TOUR
DIMENSION: 1083
TOUR_SECTION
```

1 23 55 24 2 56 64 79 …

1.2 Khởi tạo quần thể :

Quần thể ban đầu được khởi tạo bằng cách sinh ngẫu nhiên các chu trình, số lượng chu trình khởi tạo là một nửa số kích thước cá thể tối đa. Việc sinh ngẫu nhiên sử dụng hàm đột biến (sẽ nói rõ phía dưới). Số kích thước cá thể tối đa có thể tùy biến theo số đỉnh của đồ thị cần giải, sau nhiều lần chạy nhóm chọn kích thước quần thể là 100 cá thể.

1.3 Lai ghép :

Phương thức lai ghép được thực hiện dựa trên 2 cá thể đầu vào :

C1	1	3	2	4	6	9	8	7	5	10

C2	10	7	9	6	4	2	8	3	1	5

Thực hiện lai ghép 1 điểm cắt với vị trí cắt là ngẫu nhiên :

- Cắt từ điểm p đến hết chu trình của C2 đưa vào chu trình mới, lấy một ví dụ p = 5 :

Con	2	8	3	1	5					

- Xét từ đầu đến cuối chu trình 1, nạp dần các điểm chưa có trong con lai theo thứ tự duyệt ta được chu trình mới :

Con	2	8	3	1	5	4	6	9	7	10

- Tính lại chi phí cho chu trình mới sinh.

(*) Cách lai ghép này đảm bảo con lai mới là 1 chu trình thỏa mãn.

Cài đặt code :

```
private Circle hybridize(Circle c1, Circle c2) {
     Circle child = new Circle(c1.getLength());
     Random rand = new Random();
     int p = rand.nextInt(child.length - 1);
     int i = 0, j = 0, k = 0;

     for (i = p; i < child.length; i++)
     {
          child.vertex[j] = c2.vertex[i];
          j++;
     }

     for (i = 0; i < child.length; i++)
     {
          for (j = 0; j < child.length - p; j++)
          {
               if (c1.vertex[i] == child.vertex[j])
                    break;
               else
                    if (j == child.length - p - 1)
                    {
                         k++;
                         child.vertex[j + k] = c1.vertex[i];
                    }
          }
     }
```

```
    return child;
}
```

1.4 Đột biến :

Phương thức đột biến được thực hiện dựa trên 1 cá thể đầu vào :

C1	1	3	2	4	6	9	8	7	5	10

Thực hiện đột biến bằng tráo đổi các điểm trên gen cho nhau. Số lần tráo đổi được sinh ngẫu nhiên từ trong khoảng 5% chiều dài chu trình (tức là có tối đa 10% vị trí trên 1 gen có thể bị đột biến), vị trí điểm tráo cũng được sinh ngẫu nhiên trong quá trình chạy.

Ví dụ với đột biến C1 bằng tráo đổi 2 lần : tráo 3 và 9, tráo 4 và 10. Khi đó ta được chu trình mới :

C2	1	9	2	10	6	3	8	7	5	4

(*) Cách đột biến này đảm bảo cá thể mới sinh là 1 chu trình thỏa mãn.

Cài đặt code :

```
private Circle mutate(Circle c) {
    int n = c.getLength();
    Circle nextgen = new Circle(n);
    nextgen.setCircle(n, c.getCircle());
    Random rand = new Random();
    int count = rand.nextInt(mutateCoefficient) + 1;
    int p1, p2, temp;

    while (count > 0)
    {
        p1 = rand.nextInt(n);
        p2 = rand.nextInt(n);
        temp = nextgen.vertex[p1];
        nextgen.vertex[p1] = nextgen.vertex[p2];
        nextgen.vertex[p2] = temp;
```

```
            count--;
        }

        return nextgen;
}
```

1.5 Chọn lọc tự nhiên :

Kích thước quần thể là cố định qua các thế hệ. Ở mỗi thế hệ ta lại có các cá thể mới sinh bằng lai ghép và đột biến do đó cần phải có sự chọn lọc để đảm bảo tính cân bằng của quần thể cũng chính là tránh các lỗi phát sinh về bộ nhớ khi kích thước quần thể quá lớn.

$$[Số\ cá\ thể\ ở\ 1\ thế\ hệ] = [Kích\ thước\ mặc\ định] + [Số\ cá\ thể\ mới\ sinh]$$

Cách thức chọn lọc cá thể được đánh giá dựa trên chi phí của mỗi chu trình. Cá thể được chọn làm lời giả cuối cùng là cá thể có chi phí nhỏ nhất trong quần thể sau một số thế hệ tiến hóa.

Ban đầu toàn bộ quần thể sẽ được sắp xếp tăng dần về chi phí và chỉ giữ lại những cá thể thích nghi nhất (có chi phí nhỏ nhất). Tuy nhiên cách làm này có hạn chế với những bộ dữ liệu lớn. Khi số thế hệ đạt đến 1 mức nhất định, việc tìm ra chu trình nhỏ hơn ngày càng khó khăn cho nên tập cá thể trong quần thể gần như không biến đổi, điều này làm giảm sự đa dạng nguồn gen cho tiến hóa ở các thế hệ sau.

Do vậy nhóm đưa ra cách chọn lọc tự nhiên như sau :

- Sắp xếp quần thể theo chi phí tăng dần.
- Lựa chọn ngẫu nhiên 1 chỉ số : a $(0 < a < 1)$
- Loại cá thể thứ a $[Kích\ thước\ mặc\ định]$ kém thích nghi nhất từ $[Kích\ thước\ mặc\ định]$ cá thể đứng đầu danh sách quần thể.
- Loại đến khi số cá thể còn lại bằng kích thước mặc định.

Cách thức chọn lọc này đảm bảo sự phong phú của quần thể sau nhiều thế hệ. Bằng cài đặt cụ thể, cách thức chọn lọc này đem lại ***kết quả tối ưu hơn*** rất nhiều so với cách chọn lọc trước với một số bộ dữ liệu.

(*) Tối ưu Chọn lọc tự nhiên : Cách thức chọn lọc cần đến việc sắp xếp các cá thể theo chi phí. Việc sắp xếp là rất tốn kém khi quần thể có kích thước lớn. Giải thuật sắp xếp đã được sử dụng là Selection Sort (độ phức tạp $\Theta(n^2)$). Ban đầu, ở mỗi bước chọn lọc đều sắp xếp lại toàn bộ quần thể. Tuy nhiên có thể chỉ cần thực hiện sắp xếp [$Kích\ thước\ mặc\ định$] cá thể đầu tiên để giảm bớt số lượng phép so sánh và đổi chỗ.

```
private void sortList() {      // Selection sort chưa tối ưu
     int i = 0, j = 0, min;
     Circle temp;
     for (i = 0; i < population.size() - 1; i++)
     {
          min = i;
          for (j = i + 1; j < population.size(); j++)
          {
               if (population.get(j).cost < population.get(min).cost)
                    min = j;
          }
          temp = population.get(i);
          population.add(i, population.get(min));
          population.remove(i + 1);
          population.remove(min);
          population.add(min, temp);
     }
}
```

Tối ưu thủ tục sắp xếp trên bằng thay vòng lặp thứ nhất :

- Từ : `for (i = 0; i < population.size() - 1; i++)`
- Thành : `for (i = 0; i < maxPopulation - 1; i++)`

`population.size()` là [*Số cá thể ở 1 thế hệ*] còn `maxPopulation` là [*Kích thước mặc định*].

Cải tiến này đem lại ***hiệu năng tốt hơn*** cho giải thuật về thời gian chạy. Tuy nhiên so sánh trong 1 số trường hợp, cách chọn lọc này cho kết quả tồi hơn khoảng 5%.

1.6 Tiến hóa :

Với quần thể khởi tạo ban đầu, chúng được tiến hóa một cách ngẫu nhiên và thích nghi với điều kiện chọn lọc, các cá thể kém thích nghi sẽ bị loại thải và cá thể tốt nhất được chọn làm lời giải.

Việc tiến hóa diễn ra qua một số thế hệ, ở mỗi thế hệ ta thực hiện lai ghép và đột biến ngẫu nhiên trên toàn quần thể.

- Lai ghép : ngẫu nhiên trong 50% số cá thể đứng đầu tiên của quần thể (Lựa chọn cha mẹ ngẫu nhiên).
- Đột biến cho toàn bộ quần thể 100% (tuy điều này có vẻ trái tự nhiên nhưng việc tìm ra nguồn gen mới cần ưu tiên hơn hết).

Sau một số thế hệ định trước (10,000 đến 10,000,000) chương trình sẽ dừng và xuất ra kết quả.

(*) Cải tiến đa luồng : Để tận dụng hiệu năng các bộ vi xử lý đa lõi giải thuật được thiết kế với 2 luồng chạy song song. Mỗi luồng chạy độc lập, thực hiện tiến hóa cho 1 quần thể riêng biệt. Kết trả về lựa chọn từ kết quả tốt hơn.

2. Giới thiệu về chương trình :

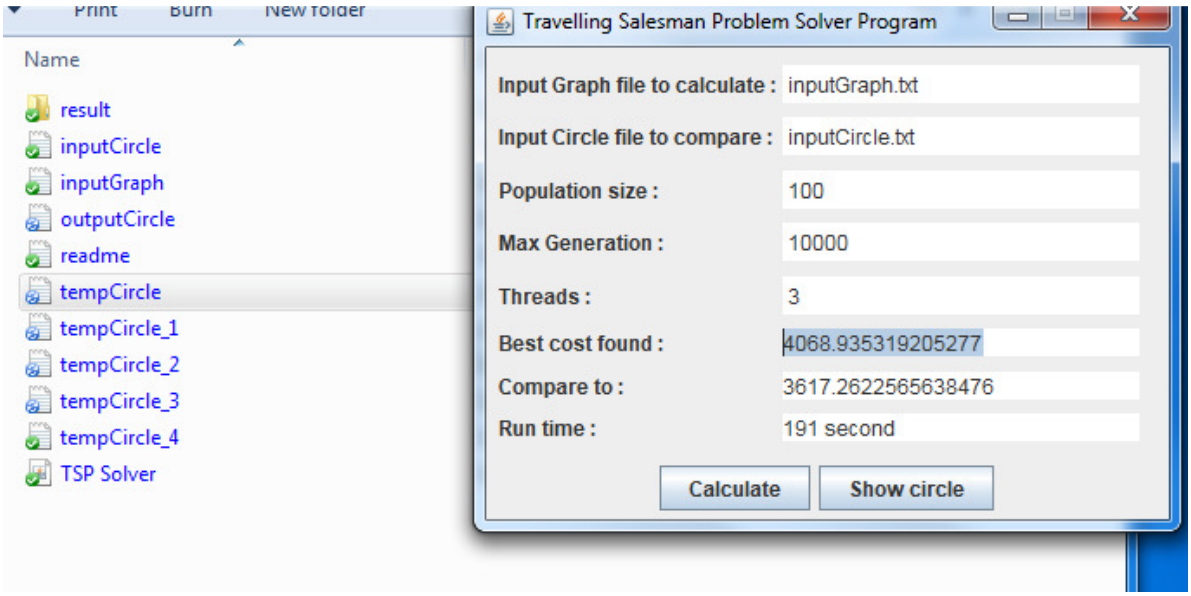


Figure 4 : Giao diện chương trình

Chương trình với giao diện đơn giản gồm 1 của sổ nhập thông tin và hiện kết quả. Mặc định chương trình sẽ đọc dữ liệu trong các tập tin đi kèm : đồ thị đầu vào inputGraph.txt và 1 chu trình mẫu để so sánh kết quả inputCircle.txt. Dữ liệu trong các tập tin phải ở định dạng chuẩn như đã đề cập trong các mục mã hóa đồ thị, mã hóa chu trình. Nếu thông tin không đúng định dạnh, chương trình sẽ không chạy.

Ngoài ra chương trình cung cấp thêm 1 số thiết lập thông số về số lượng cá thể trong quần thể (Population size) và số lượng thế hệ tiến hóa (Max Generation). Số liệu mặc định ban đầu là 100 cá thể và 10.000 thế hệ. Để giới hạn các lỗi về bộ nhớ và hạn chế thời gian chạy quá lâu, chương trình đặt sẵn số lượng cá thể lớn nhất là 1.000 và số thế hệ là 10.000.000 (nếu nhập lớn hơn sẽ tự động sử dụng giá trị lớn nhất).

Một thông số nữa cần được nhập vào đó là Threads : thông số thiết lập cho số luồng chạy song song của chương trình (số quần thể cùng tiến hóa cùng lúc).

Thông số này mặc định thiết lập là 2 và cho phép thiết lập trong khoảng từ 1 đến 4. Thiết lập đầy đủ các thông số, bấm calculate để thực hiện tính toán.

Kết quả trả về gồm chi phí cho đường đi tốt nhất mà chương trình tính toán được và chi phí tính toán từ chu trình mẫu nhập vào (có thể không cần nhập chu trình đầu vào). Ngoài ra chương trình hiển thị thời gian chạy theo giây và toàn bộ chi tiết về chu trình tính được được lưu trong tập tin outputCircle.txt, bấm nút show circle để hiện xem nội dung chu trình.

Trong quá trình tính toán, chương trình sử dụng các file tạm để lưu kết quả tính toán trung gian và kết quả tốt nhất sau các lần chạy. Khi khởi chạy nếu nhập sẵn 1 chu trình vào tập tin tempCircle.txt tương ứng với đồ thị thì chương trình có thể sẽ nhanh chóng tìm được kết quả tốt hơn.

3. Kết quả chạy các bộ dữ liệu chuẩn :

Sử dụng các bộ dữ liệu chuẩn cho bài toán TSP được tải về từ trang http://www.tsp.gatech.edu/

Dữ liệu trong mỗi bộ bao gồm thông tin về tên bộ dữ liệu, số đỉnh, dạng đồ thị và danh sách đỉnh cùng với tọa độ mỗi đỉnh.

Nhóm lựa chọn các bộ dữ liệu sau để thử nghiệm giải thuật :

Tên bộ dữ liệu	Số đỉnh	Chi phí của lời giải tối ưu đã tìm được
wi29.tsp	29	27603
qa194.tsp	194	9352
xit1083.tsp	1083	3617,26/3558

Giải thuật được cài đặt bằng ngôn ngữ **java (jdk 7u9 - win64)**

Cấu hình máy tính chạy giải thuật :

Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz
RAM DDR3 4GB bus **1600MHz**
OS Windows 7 x64

Trong quá trình chạy sử dụng thiết lập số cá thể tối đa là 100 và số thế hệ lần lượt ở các mức 10.000, 50,000, 100.000, 500.000, 1.000.000, 5.000.000 tùy bộ dữ liệu. Mỗi mức thiết lập cho chạy 10 lần và lấy ra giá trị nhỏ nhất để so sánh với chi phí tối ưu mẫu (lấy từ bộ dữ liệu chuẩn). Chi tiết về kết quả các lần chạy có trong Phụ lục đi kèm báo cáo này.

Sau đây là thống kê kết quả chạy chương trình với một số bộ dữ liệu :

3.1 Bộ dữ liệu wi29.tsp :

- Tên bộ dữ liệu wi29.tsp (Western Sahara)
- Kích thước đồ thị : 29 đỉnh
- Chi phí chu trình tối ưu : 27603

Kết quả chạy :

Pop size	Max Gen	Min cost	%	Time	Optimal found
100	10.000	27601,17	99,99	59	Yes
100	500.000	27601,17	99,99	903	Yes

(*) Có thể do kết quả tối ưu được cung cấp theo bộ dữ liệu khi tính toán đã làm tròn nên có sai số.

Kết luận : tìm thấy chu trình tối ưu.

3.2 Bộ dữ liệu qa194.tsp:

- Tên bộ dữ liệu qa194.tsp (Quatar)
- Kích thước đồ thị : 194 đỉnh

- Chi phí chu trình tối ưu : 9352

Kết quả chạy :

Pop size	Max Gen	Min cost	%	Time	Optimal found
100	1000	11465,45	122,6	17	No
100	10000	10147.12	108,5	192	No
100	100.000	10137,51	108,4	1783	No

Kết luận : chưa tìm thấy chu trình tối ưu

3.3 Bộ dữ liệu xit1083.tsp:

- Tên bộ dữ liệu xit1083.tsp (VLSI - XIT1083 – Vi mạch tích hợp)
- Kích thước đồ thị : 1083 đỉnh
- Chi phí chu trình tối ưu : 3617,26/3558

Kết quả chạy :

Pop size	Max Gen	Min cost	%	Time	Optimal found
100	1000	10774,26	297,86	80	No
100	10000	8694,27	240,36	241	No

Kết luận: chưa tìm thấy chu trình tối ưu

4. Đánh giá giải thuật và các cải tiến trong tương lai:

Giải thuật đã đề xuất đáp ứng cơ bản các bước trong giải thuật di truyền. Kết quả chạy giải thuật cho kết quả tối ưu trong các trường hợp số đỉnh nhỏ dưới 100 và đưa ra kết quả khá tiệm cận với các trường hợp số đỉnh khoảng 1000. Còn với những trường hợp số đỉnh lớn hơn 1000 giải thuật vẫn chưa tìm ra được lời giải.

Giải thuật di truyền đã đề xuất phần lớn vẫn phụ thuộc vào sự "may mắn" đề tìm ra kết quả. Do đó để tìm được kết quả tối ưu với các trường hợp số đỉnh lớn là rất hạn chế. Trong tương lai, nhóm sẽ tìm hiểu và cài đặt các giải thuật heuristic giúp tìm ra được lời giải tối ưu với đồ thị có kích thước lớn hơn, cùng với đó là các cải tiến về giải thuật GA như quần thể động, đa quần thể tương tác…

TỔNG KẾT

Báo cáo đã làm rõ các khái niệm về giải thuật di truyền và các bước thực hiện khi áp dụng vào giải quyết bài toán người du lịch. Kết quả giải thuật đã cài đặt vẫn còn nhiều hạn chế khi thời gian giải quyết các bộ dữ liệu lớn hơn cỡ 10~20 nghìn đỉnh. Trong tương lai nhóm sẽ tiếp tục cải tiến thuật toán bằng các giải thuật heuristic hỗ trợ tìm đường, cải tiến giải thuật di truyền bằng cơ chế đa quần thể có tương tác.

TÀI LIỆU THAM KHẢO

[1] Phạm Văn Hải - *Slide bài giảng Trí tuệ nhân tạo.*

[2] Nguyễn Đình Thúc - *Lập trình tiến hóa.*

[3] Wikipedia.
http://vi.wikipedia.org/wiki/B%C3%A0i_to%C3%A1n_ng%C6%B0%E1%BB%9Di_b%C3%A1n_h%C3%A0ng.

[4] Các tài liệu liên quan và các bộ dữ liệu mẫu tại http://www.tsp.gatech.edu/index.html.

PHỤ LỤC

KẾT QUẢ CHẠY GIẢI THUẬT TRÊN CÁC BỘ DỮ LIỆU

1. Bộ dữ liệu wi29.tsp :

Bộ dữ liệu wi29.tsp			**Số đỉnh : 29**		**Chi phí chu trình tối ưu : 27603**				**Chi phí chu trình tốt nhất : 27601,17**				**Tìm thấy chu trình tối ưu**		
Pop size	**Max Gen**	**Re_01**	**Re_02**	**Re_03**	**Re_04**	**Re_05**	**Re_06**	**Re_07**	**Re_08**	**Re_09**	**Re_10**	**Average run time**	**Min cost**	**Optimal cost**	**Min/opt %**
100	10000	27748,71	27748,71	27748,71	27748,71	27748,71	27748,71	27601,17	27601,17	27601,17	27601,17	59	27601,17	27603	99,99
100	100000	27601,17	27601,17	27601,17	27601,17	27601,17	27601,17	27601,17	27601,17	27601,17	27601,17	903	27601,17	27603	99,99

2. Bộ dữ liệu qa194.tsp

Bộ dữ liệu qa194.tsp			**Số đỉnh : 194**		**Chi phí chu trình tối ưu : 9352**				**Chi phí chu trình tốt nhất :**				**Chưa tìm thấy chu trình tối ưu**		
Pop size	**Max Gen**	**Re_01**	**Re_02**	**Re_03**	**Re_04**	**Re_05**	**Re_06**	**Re_07**	**Re_08**	**Re_09**	**Re_10**	**Average run time**	**Min cost**	**Optimal cost**	**Min/opt %**

100	1000	15000,60	14800,71	13448,23	13098,30	12495,81	12207,76	12048,84	11747,44	11505,10	11465,45	17	11465,45	9352	122,6
100	10000	11265,86	11158,49	11104,15	10738,32	10585,48	10483,64	10188,88	10147.12	10147.12	10147.12	192	10147.12	9352	108,5
100	100000	10147.12	10137,51	10137,51	10137,51	10137,51	10137,51	10137,51	10137,51	10137,51	10137,51	1783	10137,51	9352	108,4

3. Bộ dữ liệu xit1083.tsp :

Bộ dữ liệu xit1083.tsp			**Số đỉnh : 1083**		**Chi phí chu trình tối ưu : 3617.26**				**Chi phí chu trình tốt nhất :**				**Chưa tìm thấy chu trình tối ưu**		
Pop size	**Max Gen**	**Re_01**	**Re_02**	**Re_03**	**Re_04**	**Re_05**	**Re_06**	**Re_07**	**Re_08**	**Re_09**	**Re_10**	**Average run time**	**Min cost**	**Optimal cost**	**Min/opt %**
100	1000	14774,42	13064.93	12304.03	11987.84	11658.09	11387.66	11176.63	10957.42	10899,42	10774.26	80	10774.26	3617,26	297,86
100	10000	9599.92	9526.40	9374.46	9281,23	9119,28	9063,45	9005,01	8694,27	8914,20	8859,74	841	8694,27	3617,26	240,36