BỘ GIÁO DỤC ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

**---🙠**🕮**🙢---**



**THỰC TẬP CƠ SỞ**

**ỨNG DUNG GIẢI THUẬT DUY TRUYỀN ĐỂ GIẢI QUYẾT MỘT SỐ BÀI TOÁN**

**Giảng viên hướng dẫn: ThS. Nguyễn Thị Hương Lý**

**Sinh viên thực hiện: Trương Ngọc Tuấn**

**Mã số sinh viên: 62132542**

Nha Trang 12/2022

# LỜI NÓI ĐẦU

**Lời cảm ơn:**

Lời đầu tiên, cho phép em gửi lời cảm ơn sâu sắc và chân thành nhất đến Cô Hương Lý đã hỗ trợ em thực hiện đề tài tự chọn. Sự quan tâm và giúp đỡ của quý Cô là nguồn động viên rất lớn giúp em hoàn thành tốt đề tài này.

Với lòng biết ơn sâu sắc nhất, em xin gửi đến quý Thầy/Cô ở Khoa Công Nghệ Thông Tin đã truyền đạt vốn kiến thức quý báu cho chúng em trong suốt thời gian học tập tại trường..

Bước đầu đi vào thực tế của em còn nhiều hạn chế và bỡ ngỡ nên không tránh khỏi những thiếu sót, em rất mong nhận được những ý kiến đóng góp quý báu của quý Thầy/Cô để kiến thức của em trong bài báo cáo này được hoàn thiện hơn đồng thời có điều kiện bổ sung, nâng cao ý thức của mình.

Em xin chân thành cảm ơn!

**Mục tiêu cần đạt được:**

Giới thiệu về khái niệm về giải thuật di truyền và sử ứng dụng vào giải quyết bài toán cái túi và bài toán người du lịch.

**MỤC LỤC**

[LỜI NÓI ĐẦU 1](#_Toc123619044)

[Chương 1: TỔNG QUAN ĐỀ TÀI 3](#_Toc123619045)

[**I.** **LÝ DO CHỌN ĐỀ TÀI** 3](#_Toc123619046)

[**II.** **MÔI TRƯỜNG CÀI ĐẶT** 3](#_Toc123619047)

[**III.** **PHƯƠNG PHÁP NGHIÊN CỨU** 3](#_Toc123619048)

[Chương 2: CƠ SỞ LÝ THUYẾT 4](#_Toc123619049)

[**I.** **ĐỊNH NGHĨA** 4](#_Toc123619050)

[1. Di truyền 4](#_Toc123619051)

[2. Tiến hoá 4](#_Toc123619052)

[3. Giải thuật di truyền 5](#_Toc123619053)

[**II.** **Các bước của giải thuật di truyền** 5](#_Toc123619054)

[1. Sơ đồ thuật toán 5](#_Toc123619055)

[2. Mô tả cách hoạt động của các toán tử di truyền 7](#_Toc123619056)

[Chương 3: GIỚI THIỆU BÀI TOÁN 8](#_Toc123619057)

[I. **BÀI TOÁN CÁI TÚI** 8](#_Toc123619058)

[1. Phát biểu bài toán 8](#_Toc123619059)

[II. **BÀI TOÁN NGƯỜI DU LỊCH** 9](#_Toc123619060)

[2. Phát biểu bài toán 9](#_Toc123619061)

[Chương 4: KẾT QUẢ NGHIÊN CỨU 10](#_Toc123619062)

[**I.** **CÀI ĐẶT BÀI TOÁN CÁI TÚI** 10](#_Toc123619063)

[**1.** **Giá trị đầu vào** 10](#_Toc123619064)

[**2.** **Mã hoá bài toán** 10](#_Toc123619065)

[**3.** **Lai ghép** 11](#_Toc123619066)

[**4.** **Đột biến** 12](#_Toc123619067)

[**5.** **Tiến hoá** 13](#_Toc123619068)

[**6.** **Kết luận** 14](#_Toc123619069)

[**II.** **CÀI ĐẶT BÀI TOÁN NGƯỜI DU LỊCH** 14](#_Toc123619070)

[**1.** **Dữ liệu đầu vào** 14](#_Toc123619071)

[**2.** **Mã hoá bài toán** 16](#_Toc123619072)

[**3.** **Lai ghép** 18](#_Toc123619073)

[**4.** **Đột biến** 20](#_Toc123619074)

[**5.** **Tiến hoá** 20](#_Toc123619075)

[**6.** **Nhận xét** 21](#_Toc123619076)

[Chương 5: KẾT LUẬN 22](#_Toc123619077)

[TÀI LIỆU THAM KHẢO 23](#_Toc123619078)

# Chương 1: TỔNG QUAN ĐỀ TÀI

1. **LÝ DO CHỌN ĐỀ TÀI**

Hiện nay máy tính đã giúp con người rất nhiều bài toán. Mặt dù vậy vẫn có một số lớn các bài toán thú vị và thuật toán tốt nhất chỉ giải quyết bài toán một lời giải tương đối, ở đây, các bài toán tối ưu tổ hợp là điển hình.

Bài học này trình bày nội dung bài toán tối ưu tổ hợp là bài toán chỉ quan tâm đến một cấu hình “tốt nhất” theo một nghĩa nào đấy. Đây là bài toán có nhiều ứng dụng trong thực tiễn.

Trong khi đó giải thuật di truyền cũng như tiết hoá dựa trên khái niệm cho rằng quá trình tiến hoá tự nhiên là hoàn hảo nhất, hợp lý nhất và tự nó đã mang tính tối ưu. Sự tối ưu đó đươc thể hiện ở chổ thế hệ sau bao giờ cũng phát triển tốt hơn thế hệ trước đó.

Việc áp dụng giải thuật di truyền để giải quyết hai bài toán người du lịch và bài toán cái túi thay cho phương pháp truyền thống như nhánh cận (để giải quyết bài toán người du lịch) hay giải thuật tham lam (để giải quyết bài toán cái túi) giúp mang lại những kết quả mới mẻ và biết đâu là tốt hơn các phương pháp truyền thống trước đó.

1. **MÔI TRƯỜNG CÀI ĐẶT**

Sử dụng ngôn ngữ lập trình C++ để cài đặt thuật toán di truyền giải quyết hai bài toán nêu trên.

1. **PHƯƠNG PHÁP NGHIÊN CỨU**

Nêu lên cớ sở lý tuyết của thuật toán di truyền, nên lên vấn đề của hai bài toán thuộc dạng tối ưu tổ hợp.

Sử dụng ngôn ngữ C++ và giải thuật di truyền để giải quyết hai bài toán trên .

# Chương 2: CƠ SỞ LÝ THUYẾT

## **ĐỊNH NGHĨA**

### **Di truyền**

Di truyền" là "hiện tượng chuyển những tính trạng của cha mẹ cho con cái thông qua gen của bố mẹ". Trong sinh học, di truyền chuyển những đặc trưng sinh học từ một sinh vật cha mẹ đến con cái và nó đồng nghĩa với di chuyển gen, gen thừa nhận mang thông tin sinh học hay thông tin di truyền

### **Tiến hoá**

Tiến hóa nói đến quá trình hoàn thiện, biến đổi dần để hoàn thiện hơn các bộ phận, chức năng của các sinh vật để phù hợp hơn với điều kiện sinh tốn cũng đang dần thay đổi.

Trong sinh học, tiến hóa là sự thay đổi đặc tính di truyền của một quần thể sinh học qua những thế hệ nối tiếp nhau. Các quá trình tiến hóa làm nảy sinh sự đa dạng ở mọi mức độ tổ chức sinh học bao gồm loài, các cá thể sinh vật và cả các phân tử như ADN và protein.

Tiến hóa do chọn lọc tự nhiên là một quá trình có thể suy ra từ ba thực kiện về các quần thể sinh học:

1. Nhiều cá thể con được sinh ra hơn số lượng có thể sống sót
2. Các tính trạng khác nhau giữa các cá thể, dẫn tới tỉ lệ sinh tồn và sinh sản khác nhau
3. Những sự khác biệt về đặc điểm trên là có tính di truyền.

Do đó, khi những cá thể của một quần thể chết đi, chúng được thay thế bằng những hậu duệ của thế hệ cha mẹ nhưng có thể thích nghi tốt hơn để tồn tại và sinh sôi trong môi trường mà sự chọn lọc tự nhiên diễn ra. Quá trình này tạo ra và bảo tồn những đặc điểm được cho là phù hợp hơn cho chức năng mà chúng đảm nhiệm.

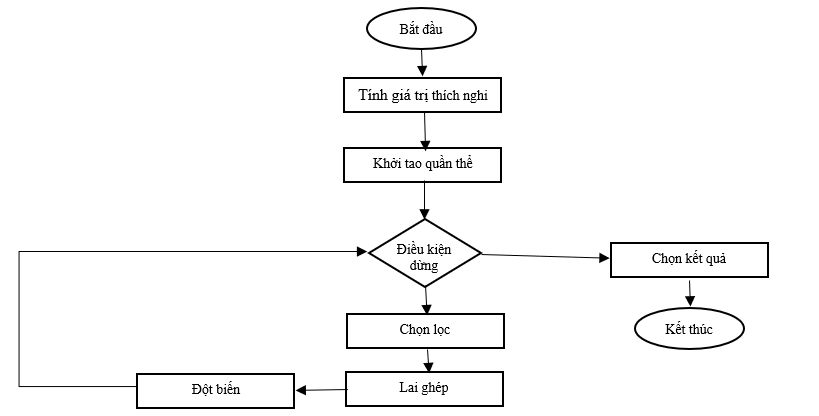
### **Giải thuật di truyền**

**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.

GA thuộc lớp các giải thuật xuất sắc nhưng lại rất khác các giải thuật ngẫu nhiên vì chúng kết hợp các phần tử tìm kiếm trực tiếp và ngẫu nhiên. Khác biệt quan trọng giữa tìm kiếm của GA và các phương pháp tìm kiếm khác là GA duy trì và xử lý một tập các lời giải, gọi là một quần thể (population). Trong GA, việc tìm kiếm giả thuyết thích hợp được bắt đầu với một quần thể, hay một tập hợp có chọn lọc ban đầu của các giả thuyết. Các cá thể của quần thể hiện tại khởi nguồn cho quần thể thế hệ kế tiếp bằng các hoạt động lai ghép và đột biến ngẫu nhiên – được lấy mẫu sau các quá trình tiến hóa sinh học. Ở mỗi bước, các giả thuyết trong quần thể hiện tại được ước lượng liên hệ với đại lượng thích nghi, với các giả thuyết phù hợp nhất được chọn theo xác suất là các hạt giống cho việc sản sinh thế hệ kế tiếp, gọi là cá thể (individual). Cá thể nào phát triển hơn, thích ứng hơn với môi trường sẽ tồn tại và ngược lại sẽ bị đào thải. GA có thể dò tìm thế hệ mới có độ thích nghi tốt hơn. GA giải quyết các bài toán quy hoạch toán học thông qua các quá trình cơ bản: lai tạo (crossover), đột biến (mutation) và chọn lọc (selection) cho các cá thể trong quần thể. Dùng GA đòi hỏi phải xác định được: khởi tạo quần thể ban đầu, hàm đánh giá các lời giải theo mức độ thích nghi – hàm mục tiêu, các toán tử di truyền tạo hàm sinh sản.

## **Các bước của giải thuật di truyền**

1. Sơ đồ thuật toán



**Giải thích sơ đồ**

Giải thuật sẽ được thực hiện qua các bước sau:

1. **Khởi tạo quần thể**: Sinh ra ngẫu nhiên một quần thể gồm n cá thể (trong đó n là lời giải cho bài toán).
2. **Tính giá trị thích nghi**: Ước lượng độ thích nghi của mỗi cá thể.
3. **Điều kiện dừng**: Kiểm tra điều kiện để kết thúc giải thuật.
4. **Chọn lọc**: Chọn hai cá thể bố mẹ từ quần thể cũ theo độ thích nghi của chúng (cá thể có độ thích nghi càng cao thì càng có nhiều khả năng được chọn).
5. **Lai ghép**: Với một xác suất được chọn, trao đổi chéo hai cá thể bố mẹ để tạo ra một cá thể mới.
6. **Đột biến**: Với một xác suất đột biến được chọn, biến đổi cá thể mới.
7. **Chọn kết quả**: Nếu thỏa mãn điều kiện dừng thì giải thuật kết thúc và chọn được lời giải tốt nhất trong quần thể hiện tại.

Ta có thể thấy rằng, khi Điều kiện dừng chưa được thỏa mán, quần thể mới sẽ liên tục được tạo ra bằng cách lặp lại 3 bước Chọn lọc, Lai ghép và Đột biến.

GAs có 2 điều kiện dừng cơ bản:

1. Dựa trên cấu trúc nhiễm sắc thể, kiểm soát số gene được hội tụ, nếu số gene được hội tụ tại một điểm hoặc vượt quá điểm đó thì giải thuật kết thúc.
2. Dựa trên ý nghĩa đặc biệt của nhiễm sắc thể, đo sự thay đổi của giải thuật sau mỗi thế hệ, nếu thay đổi này nhỏ hơn một hằng số xác định thì giải thuật kết thúc.

### **Mô tả cách hoạt động của các toán tử di truyền**

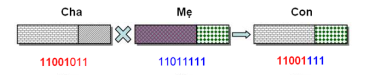
#### **Lai ghép:**

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

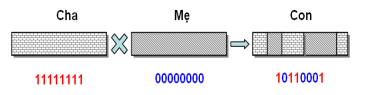
Các cặp cha mẹ được chọn ngẫu nhiên xác xuất xảy ra lai ghép với mỗi cặp được quy định từ trước.

Có nhiều cách lai ghép khác nhau:

* Lai ghép một điểm cắt, nhiều điểm cắt.



* Lai ghép nhiều đoạn

****

#### **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 sảy 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 doạn: Đảo thứ tự của một đoạn NST bất kì

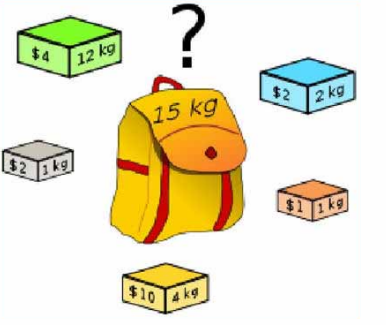
#### **Thay Thế:**

Những NST từ việc lai ghép và đột biến theo một quy tắc nào đó, thay thế cho cha mẹ để sinh ra thế hệ mới. Một số phương thức thay thế 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 các thể ưu tú nhất trong tuần thể.

# Chương 3: GIỚI THIỆU BÀI TOÁN

* 1. BÀI TOÁN CÁI TÚI



### **Phát biểu bài toán**

Bài toán cái túi được phát biểu như sau:

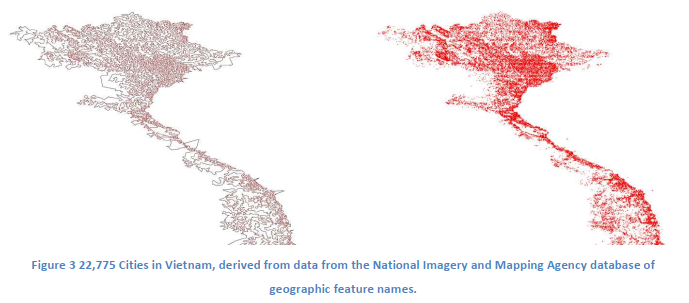
“Có n đồ vật (đánh số từ 1 đến n). Với mỗi đồ vật i, ta biết pi, vi lần lượt là các trọng lượng và giá trị của vật đó (i = 1, 2, ..., n). Giả thiết có một cái túi, sức chứa không quá w đơn vị trọng lượng. Hãy tìm một phương án chọn đồ vật bỏ vào túi để có thể mang đi được sao cho tổng giá trị các vật được mang là lớn nhất”.

Một phương án chọn đồ vật là một tập con của tập {1, 2, ..., n}, vì thế có thể biểu diễn mỗi phương án như thế như một dãy nhị phân X = (x1, x2, ..., xn), trong đó xi = 1 khi và chỉ khi vật i được chọn (i = 1, 2, ..., n). Tổng trọng lượng của các vật được mang theo phương án này là p1x1 + p2x2 + ... + pnxn. Điều kiện các vật được chọn mang đi được là điều kiện tổng này không vượt quá w (sức chứa của cái túi). Tổng giá trị các vật được mang theo phương án X là v1x1 + v2x2 + ... + vnxn. Từ đó ta nhận được mô hình toán học của bài toán cái túi như sau:

Tìm X∈ D : f (X) ∈ → max

trong đó D là tập hợp các dãy nhị phân X = (x1, x2, ..., xn) thỏa mãn bất đẳng thức p1x1 + p2x2 + ... + pnxn ≤ w và f(X) = v1x1 + v2x2 + ... + vnxn.

bài toán cái túi là thí dụ cho những bài toán tối ưu với mục tiêu là hiệu quả.

* 1. BÀI TOÁN NGƯỜI DU LỊCH

### **Phát biểu bài toán**

Bài toán người du lịch được phát biểu như sau:

“Có n thành phố (đánh số từ 1 đến n). Một người du lịch, xuất phát từ thành thành phố s, muốn đi thăm tất cả các thành phố khác, mỗi thành phố đúng một lần, rồi lại quay về nơi xuất phát. Giả thiết biết chi phí đi từ thành phố i đến thành phố j là c(i, j), 1 ≤ i, j ≤ n. Hãy tìm một hành trình cho người du lịch sao cho chi phí của hành trình này là nhỏ nhất”.

Mỗi hành trình của người du lịch được biểu diễn bằng một hoán vị X = (x1, x2, ..., xn) của {1, 2, ..., n} với x1 = s (hoán vị này biểu diễn hành trình x1→ x2→...→ xn−1→ xn→ x1). Chi phí của hành trình X được tính bằng công thức c(x1, x2) + c(x2, x3) +...+ c (xn, x1). Như thế, mô hình toán học của bài toán người du lịch là:

Tìm X∈D: f(X) min ∈ → min

trong đó D là tập các hoán vị X = (x1, x2, ..., xn) của {1, 2, ..., n} có x1 = s (cho trước) và f(X) = c(x1, x2) + c (x2, x3) +...+ c(xn, x1).

Bài toán người du lịch là thí dụ cho những bài toán tối ưu với mục tiêu là chi phí.

# Chương 4: KẾT QUẢ NGHIÊN CỨU

1. **CÀI ĐẶT BÀI TOÁN CÁI TÚI**
   1. **Giá trị đầu vào**

* Nhập vào maximum weight là trọng lượng của chiếc túi.
* Nhập number of items là số lượng đồ đang có và lần lượt nhập giá trị và trọng lượng của đồ vật.
* Nhập vào số lượng quần thể, kích cỡ của quần thể và tỉ lệ đột biến.

|  |
| --- |
| std::cout << "Enter the maximum weight: ";  std::cin >> max\_weight;  std::cout << "Enter the number of items: ";  std::cin >> num\_items;  // Initialize the items and the population  for (int i = 0; i < num\_items; i++)  {  Item item;  std::cout << "Enter the value and weight of each item: " << i << std::endl;  std::cin >> item.value;  std::cin >> item.weight;  items.push\_back(item);  }  std::cout << "Enter the population size: ";  std::cin >> population\_size;  std::cout << "Enter the number generation: ";  std::cin >> population\_size;  std::cout << "\nmutation rate (vd: 0.1 = 10%): ";  std::cin >> mutationRate; |

* 1. **Mã hoá bài toán**
* Mã hoá một vật phẩm và giải pháp:

Mỗi đồ vật được tính là một Item với 2 giá trị là value và weight, Mỗi giải pháp được mã hoá thành 1 Solution với một mảng giá trị 0 và 1 đạt diện cho chuỗi gen giải pháp đó có để vật phẩm thứ i vào hay không.

|  |
| --- |
| struct Item  {  int value;  int weight;  };  struct Solution  {  std::vector<int> genes;  int fitness;  }; |

* Khởi tạo quần thể các phương pháp và tính toán độ thích hợp cho mỗi cá thể trong quần thể

|  |
| --- |
| for (int i = 0; i < population\_size; i++)  {  for (int j = 0; j < num\_items; j++)  {  population[i].genes.push\_back(std::rand() % 2);  }  population[i].fitness = calcFitness(population[i], items, num\_items, max\_weight);  } |

* 1. **Lai ghép**
* Chọn nhiễm sắc thể lớn hơn một số ngẫu nhiên nào đó để làm bố mẹ

|  |
| --- |
| Solution selectParent(std::vector<Solution> &population, int population\_size)  {  int maxFitness = 0;  for (int i = 0; i < population\_size; i++)  {  maxFitness = std::max(maxFitness, population[i].fitness);  }  int fitnessThreshold = std::rand() % maxFitness;  for (int i = 0; i < population\_size; i++)  {  if (population[i].fitness >= fitnessThreshold)  {  return population[i];  }  }  return population[0];  } |

* Lai ghép hai nhiễm sắc thể bố mẹ với nhau với một điểm ramdom trên độ dài đoạn gen

|  |
| --- |
| Solution crossover(const Solution &parent1, const Solution &parent2, int num\_items)  {  Solution offspring;  int crossoverPoint = std::rand() % num\_items;  for (int i = 0; i < crossoverPoint; i++)  {  offspring.genes.push\_back(parent1.genes[i]);  }  for (int i = crossoverPoint; i < num\_items; i++)  {  offspring.genes.push\_back(parent2.genes[i]);  }  return offspring;  } |

* 1. **Đột biến**
* Hàm Tạo ra xác xuất đột biến từ 0 đến 1

|  |
| --- |
| bool ShouldMutate(float probability)  {  // Generate a random number between 0 and 1.  float randomNumber = static\_cast<float>(rand()) / static\_cast<float>(RAND\_MAX);  // Return true if the random number is less than the probability value, false otherwise.  return randomNumber < probability;  } |

* Đột biến:

Thay đổi nhẫu nhiên 1 nst của một cá thể thành 0 hoặc 1

|  |
| --- |
| void mutate(Solution &s, int num\_items)  {  int mutationPoint = std::rand() % num\_items;  s.genes[mutationPoint] = !s.genes[mutationPoint];  } |

* 1. **Tiến hoá**
* Lặp lại và thay thế những cá thể kém thành cá thể tốt hơn

|  |
| --- |
| while (numGenerations <generation)  {  // Select the parents  Solution parent1 = selectParent(population, population\_size);  Solution parent2 = selectParent(population, population\_size);  // Perform crossover to create the offspring  Solution offspring = crossover(parent1, parent2, num\_items);    // Perform mutation on the offspring  if (ShouldMutate(mutationRate))  {  mutate(offspring, num\_items);  }  // Evaluate the fitness of the offspring  offspring.fitness = calcFitness(offspring, items, num\_items, max\_weight);  // Replace the least fit member of the population with the offspring  int leastFitIndex = 0;  for (int i = 1; i < population\_size; i++)  {  if (population[i].fitness < population[leastFitIndex].fitness)  {  leastFitIndex = i;  }  }  population[leastFitIndex] = offspring;  numGenerations++;  } |

* 1. **Kết luận**

Thuật toán đã cho ra những kết quả có độ tốt cao có thể sử dụng thay thế thuật toán tham lam, khắc phục được việc tràn bộ nhớ do sử dụng đệ quy.

Tuy nhiên các kết quả của bài toán còn ngẫu nhiên và cần phải tạo một quần thể đủ nhiều để có thể tìm ra phương án tốt nhất, điều đó khiến thuật toán có phần chậm chạp đi.

1. **CÀI ĐẶT BÀI TOÁN NGƯỜI DU LỊCH**
   1. **Dữ liệu đầu vào**

Dữ liệu đầu vào là một bộ dữ liệu từ trang : <https://www.math.uwaterloo.ca/tsp/world/> ( cung cấp các bộ dữ liệu chuẩn quốc tế)

Vd về chuẩn dữ liệu đồ thị:

|  |  |  |
| --- | --- | --- |
| NAME : xit78 COMMENT : Bonn VLSI data set with 78 points COMMENT : Uni Bonn, Research Institute for Discrete Math COMMENT : Contributed by Andre Rohe TYPE : TSP DIMENSION : 78 EDGE\_WEIGHT\_TYPE : EUC\_2D NODE\_COORD\_SECTION | | |
| 1 | 0 | 102 |
| 2 | 0 | 105 |
| 3 | 0 | 107 |
| … | … | … |

* Đọc dữ liệu đầu vào:

dữ liệu sẽ được đọc từng dòng, khi đọc tới dòng NODE\_COORD\_SECTION sẽ bắt đầu đọc từng số, và lưu vào trong một biến kiểu point

|  |
| --- |
| struct Point  {  float x;  float y;  };  int main()  {  std::string line;  std::vector<Point> points;  while (std::getline(input\_file, line))  {  std::stringstream ss(line);  std::string token;  std::vector<std::string> tokens;  while (ss >> token)  {  tokens.push\_back(token);  }  if (tokens[0] == "NODE\_COORD\_SECTION")  {  while (std::getline(input\_file, line))  {  if (line == "EOF")  {  break;  }  std::stringstream ss(line);  int id;  float x;  float y;  ss >> id >> x >> y;  Point p{x, y};  points.push\_back(p);  }  }  } |

* Tính toán khoảng cách giữa 2 điểm:

Dữ liệu sẽ được tính toán từ hàm Distance và đổi sang ma trận kề, lưu trữ vào vetor 2 chiều distances như sau:

|  |
| --- |
| struct Point  {  float x;  float y;  };  float Distance(const Point &p1, const Point &p2)  {  int dx = p1.x - p2.x;  int dy = p1.y - p2.y;  return std::sqrt(dx \* dx + dy \* dy);  }  …  Int main ()  {  std::vector<std::vector<float>> distances(n, std::vector<float>(n));  for (int i = 0; i < n; i++)  {  for (int j = 0; j < n; j++)  {  distances[i][j] = Distance(points[i], points[j]);  }  }  … |

* Tiết theo nhập vào số lượng quần thể, kích cỡ của quần thể và tỉ lệ đột biến cũng như cách hiển thị bài toán
  1. **Mã hoá bài toán**

Ta mã hoá mỗi chuyến đi thành một cá thể như sau:

* Mã hoá chuỗi thành phố:

Mảng cities trong struct Chromosome sẽ lưu danh sách các thành phố Trong khi đó biến fitness sẽ lưu độ dài của quãng đường đi qua các thành phố đó Như sau:

|  |
| --- |
| struct Chromosome  {  std::vector<int> cities;  float fitness;  }; |

* Khởi tạo quần thể mới với các giá trị ngẫu nhiên:

Ta sẽ tạo một quần thể ngẫu nhiên các chuyến đi bắt đầu từ thành phố [0] đi tới tất cả các thành phố khác 1 lần và quay về bằng cách thêm lần lượt các thành phố và sử dụng một hàm để sắp xếp lại các thành phố một cách ngẫu nhiên như sau:

|  |
| --- |
| void InitPopulation(std::vector<Chromosome> &population, int populationSize, int n)  {  for (int i = 0; i < populationSize; i++)  {  Chromosome c;  // The first city in the tour should always be city 0.  c.cities.push\_back(0);  // Add the remaining cities in a random order.  for (int j = 1; j < n; j++)  {  c.cities.push\_back(j);  }  std::random\_shuffle(c.cities.begin() + 1, c.cities.end());  population.push\_back(c);  }  } |

* Tính toán lại quãng đường của một các thể:

Ta có một hàm để tính toán lại tổng quãng đường đi của một cá thể dựa vào vị trí giá trị đường đi của nó trong ma trận kề như sau:

|  |
| --- |
| float CalcFitness(const Chromosome &c, std::vector<std::vector<float>> distances, int n)  {  float fitness = 0;  for (int i = 0; i < n - 1; i++)  {  fitness += distances[c.cities[i]][c.cities[i + 1]];  }  fitness += distances[c.cities[n - 1]][c.cities[0]];    return -fitness;  } |

* 1. **Lai ghép**
* Lựa chọn nhiễm sắc thể cha mẹ:

Lựa chọn ngẫu nhiên 2 nhiễm sắc thể bất kì trong quần thể

|  |
| --- |
| void Select(const std::vector<Chromosome> &population, Chromosome &c1, Chromosome &c2, int tournamentSize)  {  std::vector<Chromosome> tournament;  for (int i = 0; i < tournamentSize; i++)  {  int r = rand() % population.size();  tournament.push\_back(population[r]);  }  // Select the fittest chromosome from the tournament.  Chromosome fittest = tournament[0];  for (int i = 1; i < tournamentSize; i++)  {  if (tournament[i].fitness > fittest.fitness)  {  fittest = tournament[i];  }  }  c1 = fittest;  // Repeat the process to select the second chromosome.  tournament.clear();  for (int i = 0; i < tournamentSize; i++)  {  int r = rand() % population.size();  tournament.push\_back(population[r]);  }  fittest = tournament[0];  for (int i = 1; i < tournamentSize; i++)  {  if (tournament[i].fitness > fittest.fitness)  {  fittest = tournament[i];  }  }  c2 = fittest;  } |

* Lai ghép 2 nhiễm sắc thể được chọn:

Chọn hai nhiễm sắc thể và một điểm ngẫu nhiên ở giữa đoạn nhiễm sắc thể, để tiến hành ghép gen của các thể bố và gen của cá thể mẹ vào với nhau, từ đó tạo ra cá thể mới.

|  |
| --- |
| void Crossover(const Chromosome &c1, const Chromosome &c2, Chromosome &offspring1, Chromosome &offspring2, int n)  {  int crossoverPoint = rand() % n;  offspring1.cities.assign(c1.cities.begin(), c1.cities.begin() + crossoverPoint);  offspring2.cities.assign(c2.cities.begin(), c2.cities.begin() + crossoverPoint);  for (int i = 0; i < n; i++)  {  int city = c2.cities[i];  if (std::find(offspring1.cities.begin(), offspring1.cities.end(), city) == offspring1.cities.end())  {  offspring1.cities.push\_back(city);  }  }  for (int i = 0; i < n; i++)  {  int city = c1.cities[i];  if (std::find(offspring2.cities.begin(), offspring2.cities.end(), city) == offspring2.cities.end())  {  offspring2.cities.push\_back(city);  }  }  } |

* 1. **Đột biến**
* Tỉ lệ đột biến:

Nhận vào một số từ 0.0 đến 1 và trả ra xác xuất sảy ra:

|  |
| --- |
| bool ShouldMutate(float probability)  {  // Generate a random number between 0 and 1.  float randomNumber = static\_cast<float>(rand()) / static\_cast<float>(RAND\_MAX);  // Return true if the random number is less than the probability value, false otherwise.  return randomNumber < probability;  } |

* Đột biến cá thể:

Hoán đổi vị trí của 2 thành phố và tạo ra một chuyến đi mới (2 thành phố phải khác thành phố đầu tiên) :

|  |
| --- |
| void Mutate(Chromosome &c)  {  // Choose two random indices to swap.  int i = rand() % c.cities.size();  int j = rand() % c.cities.size();  // Only swap the cities if neither of the chosen indices is 0.  while (i == 0 || j == 0)  {  i = rand() % c.cities.size();  j = rand() % c.cities.size();  }  std::swap(c.cities[i], c.cities[j]);  } |

* 1. **Tiến hoá**
* Sắp xếp lại các cá thể và thêm những cá thể mới:

Sắp xếp theo độ thích nghi và giữ lại ½ cá thể tốt nhất sau đó nhận các cá thể được lai ghép và đột biến đến khi đủ số lượng cá thể trong quần thể

|  |
| --- |
| std::sort(population.begin(), population.end(), [](const Chromosome &c1, const Chromosome &c2)  { return c1.fitness > c2.fitness; });  ….  std::vector<Chromosome> newPopulation;  for (int j = 0; j < eliteSize; j++)  {  newPopulation.push\_back(population[j]);  }  while (newPopulation.size() < populationSize)  {  // Select the parents using tournament selection.  Chromosome c1, c2;  Select(population, c1, c2, tournamentSize);  // Perform crossover to produce offspring.  Chromosome offspring1, offspring2;  Crossover(c1, c2, offspring1, offspring2, n);  // Mutate the offspring.  if (ShouldMutate(mutationRate))  {  Mutate(offspring1);  Mutate(offspring2);  }  // Add the offspring to the new population.  newPopulation.push\_back(offspring1);  if (newPopulation.size() < populationSize)  {  newPopulation.push\_back(offspring2);  }  } |

* 1. **Nhận xét**

Thuật toán đã áp dụng và tìm ra lời giải tương đối tốt, một số trườn hợp ghi nhận lời giải tốt hơn phương pháp nearest neighbor.

Tuy nhiên do việc lai ghép hai cá thể để tạo ra cá thể mới chưa tối ưu tốt, nên tốc độ sử lý của bài toán còn hạn chế.

# Chương 5: KẾT LUẬN

Các bài toán tổ hợp luôn là thách thức với giới khoa học, vì số lượng nghiệm rất lớn và chúng ta chỉ sử dụng các thuật toán để tìm đủ tốt chứa chưa phải là tối ưu nhất.

Việc ứng dụng thuật toán di truyền vào những bài toán tổ hợp tương đốt hiệu quả để tìm ra một nghiệm tối ưu.

Tuy nhiên vì là một phương pháp tối ưu sự ngẫu nhiên nên kết quả phụ thuộc phần lớn vào độ lớn của quần thể và cá thể khởi tạo cũng như việc chưa tối ưu hoá các bước lai ghép, cũng khiến cho thuật toán trở nên chậm hơn so với những thuật toán khác.

**Soue code: https://github.com/trntuan/UsingTheGeneticAlgorithmGA.git**

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | http://eldata11.topica.edu.vn/HocLieu/v1.0/MAT104/Giao%20trinh/06-MAT104-Bai%204-v1.0.pdf |
| [2] | http://mit.dthu.edu.vn/uploads/TKHuong.pdf |
| [3] | https://people.sc.fsu.edu/~jburkardt/datasets/cities/cities.html |
| [4] | https://viblo.asia/p/bai-toan-cai-tui-va-nhung-ung-dung-xung-quanh-no-maGK7Nke5j2 |