Báo cáo chương trình giải bài toán TSP bằng giải thuật di truyền

Ngày 17 tháng 6 năm 2025

Mục lục

1 Giới thiệu

Chương trình này được viết bằng ngôn ngữ C++ để giải bài toán người bán hàng (TSP - Travelling Salesman Problem) sử dụng giải thuật di truyền (genetic algorithm). Chương trình tự động sinh dữ liệu 20 thành phố với tọa độ ngẫu nhiên và tìm đường đi ngắn nhất qua tất cả các thành phố đó.

2 Nguyên lý giải thuật

- Khởi tạo quần thể ban đầu gồm nhiều đường đi ngẫu nhiên.
- Xếp hạng quần thể dựa trên khoảng cách của từng đường đi.
- Chọn lọc: ưu tiên các đường đi tốt hơn (khoảng cách ngắn hơn).
- Lai ghép (crossover): kết hợp hai đường đi để tạo ra đường đi con.
- Đột biến (mutation): hoán đổi vị trí hai thành phố trong đường đi với xác suất nhỏ.
- Lặp lại quá trình trên qua nhiều thế hệ để cải thiện chất lượng lời giải.

3 Cấu trúc chương trình

- **City**: Cấu trúc lưu tọa độ x, y của một thành phố.
- **TSPSolver**: Lớp chính xử lý toàn bộ logic của giải thuật:
  - Tạo quần thể ban đầu.
  - Đánh giá và xếp hạng các cá thể.
  - Thực hiện chọn lọc, lai ghép, đột biến.
  - Sinh thế hệ mới.
  - Lặp lại qua nhiều thế hệ để tìm đường đi ngắn nhất.

4 Giải thích mã nguồn C++

- `distance(a, b)`: Tính khoảng cách Euclid giữa hai thành phố:

```
double distance(const City& a, const City& b){
return sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2));
}

‘‘‘
```

- `createRandomRoute()`: Sinh một đường đi ngẫu nhiên qua tất cả thành phố:

```
vector<int> createRandomRoute(){
    vector<int> route(cities.size());
    for (int i = 0; i < cities.size(); ++i) route[i] = i;
    random_shuffle(route.begin(), route.end());
    return route;
}
```

- `calculateTotalDistance(route)`: Tính tổng khoảng cách của một đường đi:

```
double calculateTotalDistance(const vector<int>& route){
    double total = 0.0;
    for (int i = 0; i < route.size(); ++i){
        int from = route[i];
        int to = route[(i + 1) % route.size()];
        total += distance(cities[from], cities[to]);
    }
    return total;
}
```

- `rankRoutes(population)`: Tính fitness (1 / tổng khoảng cách):

```
vector<pair<int, double>> rankRoutes(const vector<vector<int>>& population){
    vector<pair<int, double>> fitnessResults;
    for (int i = 0; i < population.size(); ++i)
        fitnessResults.emplace_back(i, 1.0 / calculateTotalDistance(population[i]
    sort(fitnessResults.begin(), fitnessResults.end(), [](auto& a, auto& b){
        return a.second > b.second;
    });
    return fitnessResults;
}
```

- `selection(popRanked)`: Chọn cá thể cho lai ghép:

```
vector<int> selection(const vector<pair<int, double>>& popRanked){
    vector<int> selectionResults;
    // Chọn elite
    for (int i = 0; i < elite_size; ++i)
        selectionResults.push_back(popRanked[i].first);
    // Xác suất lựa chọn còn lại
    double totalFitness = accumulate(...);
    // Tính xác suất cộng dồn, chọn ngẫu nhiên
    ...
    return selectionResults;
}
```

- `breed(parent1, parent2)`: Lai ghép hai cá thể:

```
vector<int> breed(const vector<int>& parent1, const vector<int>& parent2){
    ... // chọn đoạn con từ parent1
    ... // thêm phần còn thiếu từ parent2
    return child;
}
```

- `mutate(route)`: Đột biến một cá thể bằng cách hoán đổi ngẫu nhiên:

```
vector<int> mutate(const vector<int>& route){
    ... // với xác suất mutation_rate hoán đổi 2 thành phố
    return mutatedRoute;
}
```

- `nextGeneration(currentGen)`: Sinh thế hệ mới từ thế hệ hiện tại:

```
vector<vector<int>> nextGeneration(const vector<vector<int>>& currentGen){
    ... // chọn lọc, lai ghép, đột biến
    return nextGen;
}
```

- `geneticAlgorithm()`: Vòng lặp chính chạy qua nhiều thế hệ:

```
pair<vector<int>, double> geneticAlgorithm(){
    ... // lặp từ 0 đến generations
    ... // cập nhật đường đi tốt nhất
    return {bestRoute, bestDistance};
}
```

‘‘‘

5 Kết luận

Chương trình sử dụng giải thuật di truyền để tìm đường đi ngắn nhất cho bài toán TSP với số lượng thành phố nhất định. Thông qua quá trình tiến hóa qua nhiều thế hệ với chọn lọc, lai ghép và đột biến, chương trình dần cải thiện chất lượng lời giải và đưa ra đường đi tối ưu nhất tìm được.