|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| СПКм-12 | 5 | **Використання генетичного алгоритму для задачі комівояжера (TSP)** |  |  |
| Баландюх О.А. | |
| № залікової: | |
| Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні | | Викладач: | |
| Кривий Р.З. | |

**Мета роботи**: Ознайомитися з основними теоретичними відомостями, вивчити еволюційні оператори схрещування та мутації, що використовуються при розв’язуванні задач комбінаторної оптимізації.

**Завдання:** Розробити за допомогою довільної мови програмуванняпрограмне забезпечення длявирішення задачі комівояжера.

Варіант - 2

Використовуючи селекцію усіканням.

**Результати виконання програми**

**РЕЗУЛЬТАТИ ВИКОНАННЯ РОБОТИ**

/\* ----------------------- \*/

City size: 10

[6,1;7,6;8,3;7,4;6,6;9,1;4,2;8,4;9,6;5,1;]

/\* ----------------------- \*/

Initial distance: 24

Finished

Final distance: 19

Time: 48мс

Solution:

|8, 3|7, 4|9, 6|7, 6|6, 6|4, 2|5, 1|6, 1|9, 1|8, 4|

/\* ----------------------- \*/

Initial distance: 22

Finished

Final distance: 18

Time: 71мс

Solution:

|9, 1|8, 3|8, 4|7, 4|9, 6|7, 6|6, 6|4, 2|5, 1|6, 1|

/\* ----------------------- \*/

Initial distance: 23

Finished

Final distance: 18

Time: 133мс

Solution:

|7, 4|9, 6|7, 6|6, 6|8, 4|8, 3|9, 1|6, 1|5, 1|4, 2|

/\* ----------------------- \*/

City size: 20

[2,16;6,0;15,4;8,7;18,12;0,1;9,6;3,14;18,18;12,1;15,1;14,15;10,4;2,3;12,9;5,4;2,18;2,7;9,18;14,6;]

/\* ----------------------- \*/

Initial distance: 179

Finished

Final distance: 108

Time: 33мс

Solution:

|18, 18|14, 15|9, 18|2, 18|3, 14|2, 16|2, 7|0, 1|5, 4|8, 7|10, 4|15, 4|15, 1|12, 1|6, 0|2, 3|9, 6|12, 9|18, 12|14, 6|

/\* ----------------------- \*/

Initial distance: 169

Finished

Final distance: 99

Time: 170мс

Solution:

|8, 7|12, 1|6, 0|10, 4|5, 4|0, 1|2, 3|2, 7|3, 14|2, 16|2, 18|9, 18|14, 15|18, 18|18, 12|14, 6|12, 9|9, 6|15, 1|15, 4|

/\* ----------------------- \*/

Initial distance: 164

Finished

Final distance: 91

Time: 100мс

Solution:

|15, 1|12, 1|10, 4|6, 0|2, 7|2, 3|0, 1|5, 4|3, 14|2, 18|2, 16|9, 18|14, 15|18, 18|18, 12|12, 9|9, 6|8, 7|14, 6|15, 4|

/\* ----------------------- \*/

Таблиця порівняння кількості міст і популяції

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Кількість міст | 10 | | | 20 | | |
| Популяція | 20 | 100 | 200 | 20 | 100 | 200 |
| Час виконання,c | 48 | 71 | 133 | 33 | 170 | 100 |
| Довжина шляху | 19 | 18 | 18 | 108 | 99 | 91 |

**ЛІСТИНГ ПРОГРАМИ**

Лістинг файлу “Tour.java”

**public** **class** Tour {

// Зберігання проходу міст

**private** List<City> tour = **new** ArrayList<City>();

**private** **double** fitness = 0;

**private** **int** distance = 0;

**private** TourManager tourManager = **null**;

// Створює порожній тур

**public** Tour(TourManager tourManager) {

**this**.tourManager = tourManager;

**for** (**int** i = 0; i < tourManager.size(); i++) {

tour.add(**null**);

}

}

// Створення рамдомного індивідууму

**public** **void** generateIndividual() {

// Перегляд всіх відстаней міст і додавання їх до проходу

**for** (**int** cityIndex = 0; cityIndex < tourManager.size(); cityIndex++) {

setCity(cityIndex, tourManager.get(cityIndex));

}

// Випадкова зміна проходу

Collections.*shuffle*(tour);

}

// Отримання міста з проходу

**public** City getCity(**int** tourPosition) {

**return** (City) tour.get(tourPosition);

}

// Встановлює місто в певному положені проходу

**public** **void** setCity(**int** tourPosition, City city) {

tour.set(tourPosition, city);

// Якщо проходи були замінені то потрібно занулити відстань і

// пристосованість

fitness = 0;

distance = 0;

}

// Отримання пристосованості проходу

**public** **double** getFitness() {

**if** (fitness == 0) {

fitness = 1 / (**double**) getDistance();

}

**return** fitness;

}

// Отримання загальної відстанні проходу

**public** **int** getDistance() {

**if** (distance == 0) {

**int** tourDistance = 0;

// Перебір міст проходу

**for** (**int** cityIndex = 0; cityIndex < tourSize(); cityIndex++) {

// Отримання міста з якого рухаємося

City fromCity = getCity(cityIndex);

// Місто до якого рухаємося

City destinationCity;

// Перевірка чи є останнє місто проходу

**if** (cityIndex + 1 < tourSize()) {

destinationCity = getCity(cityIndex + 1);

} **else** {

destinationCity = getCity(0);

}

// Отримання дистаннції між двома містами

tourDistance += fromCity.distanceTo(destinationCity);

}

distance = tourDistance;

}

**return** distance;

}

// Отримання кількості міст в проході

**public** **int** tourSize() {

**return** tour.size();

}

// Перевірка чи прохід містить місту

**public** **boolean** containsCity(City city) {

**return** tour.contains(city);

}

@Override

**public** String toString() {

String geneString = "|";

**for** (**int** i = 0; i < tourSize(); i++) {

geneString += getCity(i) + "|";

}

**return** geneString;

}

}

Лістинг файлу “TourManager.java”

**public** **class** TourManager {

// Зберігання відстаней міст

**private** List<City> destinationCities = **new** ArrayList<City>();

// Додавання відстанні міста

**public** **void** add(City city) {

destinationCities.add(city);

}

// Отримання міста

**public** City get(**int** index) {

**return** (City) destinationCities.get(index);

}

// Отримання к-сті міст та населених пунктів

**public** **int** size() {

**return** destinationCities.size();

}

}

Лістинг файлу “Population.java”

**public** **class** Population {

// Зберігання популяції турів

**private** Tour[] tours = **null**;

// Створити популцію

**public** Population(**int** populationSize, **boolean** initialise, TourManager tourManager) {

tours = **new** Tour[populationSize];

**if** (initialise) {

// Створення індивідумів

**for** (**int** i = 0; i < populationSize(); i++) {

Tour newTour = **new** Tour(tourManager);

newTour.generateIndividual();

saveTour(i, newTour);

}

}

}

// Зберегти прохід

**public** **void** saveTour(**int** index, Tour tour) {

tours[index] = tour;

}

// Отримати прохід з популяції

**public** Tour getTour(**int** index) {

**return** tours[index];

}

// Отримати кращий прохід в популяції

**public** Tour getFittest() {

Tour fittest = tours[0];

// Переглянути індивідуми щоб знайти найкращий

**for** (**int** i = 1; i < populationSize(); i++) {

**if** (fittest.getFitness() <= getTour(i).getFitness()) {

fittest = getTour(i);

}

}

**return** fittest;

}

// Отримати розмір популяції

**public** **int** populationSize() {

**return** tours.length;

}

}

Лістинг файлу “GA.java”

**public** **class** GA {

/\* ГА параметри \*/

**private** **static** **final** **int** ***tournamentSize*** = 5;// кількість проходів для

// турнірної селекції

**private** TourManager tourManager = **null**;

**public** GA(TourManager tourManager) {

**this**.tourManager = tourManager;

}

// Розвивається населенням більше одного покоління

**public** Population evolvePopulation(Population pop) {

Population newPopulation = **new** Population(pop.populationSize(), **false**, tourManager);

**int** elitismOffset = 0;

// кросовер населення

// Цикл розміром нового населення і створення осіб поточного населення

**for** (**int** i = elitismOffset; i < newPopulation.populationSize(); i++) {

// Виберір батьків

Tour parent1 = tournamentSelection(pop);

Tour parent2 = tournamentSelection(pop);

// кросовер батьків

Tour child = crossover(parent1, parent2);

// добавити нащадка до нової популяції

newPopulation.saveTour(i, child);

}

// Провести мутацію нової популяції

**for** (**int** i = elitismOffset; i < newPopulation.populationSize(); i++) {

mutate(newPopulation.getTour(i));

}

**return** newPopulation;

}

// двохточкове впорядковуюче

**public** Tour crossover(Tour parent1, Tour parent2) {

// створити новий прохід нащадка

Tour child = **new** Tour(tourManager);

**int** p1 = (**int**) (Math.*random*() \* parent1.tourSize());

**int** p2 = (**int**) (Math.*random*() \* parent1.tourSize());

**for** (**int** i = 0; i < child.tourSize(); i++) {

**if** (i >= p1 && i <= p2) {

child.setCity(i, **null**);

} **else** {

child.setCity(i, parent1.getCity(i));

}

}

**int** n = 0;

**for** (**int** j = 0; j < parent2.tourSize(); j++) {

**boolean** t = **false**;

**for** (**int** k = 0; k < child.tourSize(); k++) {

**if** (parent2.getCity(j) == child.getCity(k)) {

t = **true**;

**break**;

}

}

**if** (t == **false**) {

child.setCity(p1 + n, parent2.getCity(j));

n = n + 1;

}

}

**return** child;

}

// класичне інвертування

**private** **void** mutate(Tour tour) {

**int** tourPos1 = (**int**) (tour.tourSize() \* Math.*random*());

**int** tourPos2 = (**int**) (tour.tourSize() \* Math.*random*());

**if** (tourPos2 < tourPos1) {

**int** q = tourPos1;

tourPos1 = tourPos2;

tourPos2 = q;

}

**for** (**int** i = 0; i <= (tourPos2 - tourPos1 + 1) / 2; i++) {

City q = tour.getCity(tourPos1 + i);

tour.setCity(tourPos1 + i, tour.getCity(tourPos2 - i));

tour.setCity(tourPos2 - i, q);

}

}

// Вибрати кандидатів проходу для кросинговера

**private** Tour tournamentSelection(Population pop) {

// Створити турнірну популяцію

Population tournament = **new** Population(***tournamentSize***, **false**,

tourManager);

// Для кожного міста в турнірі отримати рандомного кандидата проходу і

// добавити його

**for** (**int** i = 0; i < ***tournamentSize***; i++) {

**int** randomId = (**int**) (Math.*random*() \* pop.populationSize());

tournament.saveTour(i, pop.getTour(randomId));

}

// взяти найкращий прохід

Tour fittest = tournament.getFittest();

**return** fittest;

}

}

Лістинг файлу “City.java”

**public** **class** City {

**int** x;

**int** y;

// Створює місто в обраних х, у координатах

**public** City(**int** x, **int** y){

**this**.x = x;

**this**.y = y;

}

// Отримує координату х міста

**public** **int** getX(){

**return** **this**.x;

}

// Отримує координату у міста

**public** **int** getY(){

**return** **this**.y;

}

// Отримує відстань до даного міста

**public** **double** distanceTo(City city){

**int** xDistance = Math.*abs*(getX() - city.getX());

**int** yDistance = Math.*abs*(getY() - city.getY());

**double** distance = Math.*sqrt*( (xDistance\*xDistance) + (yDistance\*yDistance) );

**return** distance;

}

**public** String toString(){

**return** getX()+", "+getY();

}

}

Лістинг файлу “TSP\_GA.java”

**public** **class** TSP\_GA {

**private** TourManager tourManager = **new** TourManager();

**private** GA ga = **new** GA(tourManager);

**public** TSP\_GA(**int** x[], **int** y[], **int** populationSize) {

// Створення і додавання міста

**for** (**int** i = 0; i < x.length; i++) {

tourManager.add(**new** City(x[i], y[i]));

}

// Ініціалізація популяції

Population pop = **new** Population(populationSize, **true**, tourManager);

System.***out***.println("Initial distance: " + pop.getFittest().getDistance());

Date currentTimeBefore = **new** Date();

**long** timeBefore = currentTimeBefore.getTime();

// System.out.println("Time: " + timeBefore);

// Розвинення населеня на 100 поколінь

pop = ga.evolvePopulation(pop);

**for** (**int** i = 0; i < 100; i++) {

pop = ga.evolvePopulation(pop);

}

Date currentTimeAfter = **new** Date();

**long** timeAfter = currentTimeAfter.getTime();

// System.out.println("Time: " + timeAfter);

**long** time = timeAfter - timeBefore;

// Вивід результатів

System.***out***.println("Finished");

System.***out***.println("Final distance: " + pop.getFittest().getDistance());

System.***out***.println("Time: " + time + "мс");

System.***out***.println("Solution:");

System.***out***.println(pop.getFittest());

}

**public** **static** **void** main(String[] args) {

**int** citySize = 10;

**int**[] x = **new** **int**[citySize];

**int**[] y = **new** **int**[citySize];

Random random = **new** Random();

System.***out***.println("/\* ----------------------- \*/");

System.***out***.println("City size: " + citySize);

System.***out***.print("[");

**for** (**int** i = 0; i < citySize; i++) {

x[i] = random.nextInt(citySize);

y[i] = random.nextInt(citySize);

System.***out***.print(x[i] + "," + y[i] + ";");

}

System.***out***.println("]");

System.***out***.println("/\* ----------------------- \*/");

**new** TSP\_GA(x, y, 20);

System.***out***.println("/\* ----------------------- \*/");

**new** TSP\_GA(x, y, 100);

System.***out***.println("/\* ----------------------- \*/");

**new** TSP\_GA(x, y, 200);

citySize = 20;

x = **new** **int**[citySize];

y = **new** **int**[citySize];

System.***out***.println("/\* ----------------------- \*/");

System.***out***.println("City size: " + citySize);

System.***out***.print("[");

**for** (**int** i = 0; i < citySize; i++) {

x[i] = random.nextInt(citySize);

y[i] = random.nextInt(citySize);

System.***out***.print(x[i] + "," + y[i] + ";");

}

System.***out***.println("]");

System.***out***.println("/\* ----------------------- \*/");

**new** TSP\_GA(x, y, 20);

System.***out***.println("/\* ----------------------- \*/");

**new** TSP\_GA(x, y, 100);

System.***out***.println("/\* ----------------------- \*/");

**new** TSP\_GA(x, y, 200);

System.***out***.println("/\* ----------------------- \*/");

}

}

**Висновок:** під час виконання лабораторної роботи, я отримав практичні навички необхідні для програмної реалізації алгоритму, який дозволяє вирішувати задачу комбінаторної оптимізації. Згідно з лабораторним завданням я написав програму за допомогою мови програмування Java, що дозволяє розв’язувати різні види задач комівояжера з застосування генетичних алгоритмів.