|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| СПКс-11 | 5 | ЗАПРОГРАМУВАТИ ГА ДЛЯ  ЗАДАЧІ КОМІВОЯЖЕРА (TSP) |  |  |
| Брилюк Д.А. | |
| № залікової: 1408525 | |
| 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.

В - 2

Використовуючи селекцію усіканням.

**Результати виконання програми**

Програма запускалася 6 раз з різною кількістю міст і популяцією. Для порівняння було вибрано кількість міст 10, 20, а популяцію 20, 100,200.

Координати для 10 міст

[12,2;5,3;15,15;17,3;8,16;4,13;6,14;6,19;5,14;2,9;]

Координати для 20 міст

[7,20;13,8;4,13;5,18;12,15;8,15;6,7;13,17;1,13;11,6;16,3;20,12;16,12;6,19;13,5;11,10;13,12;7,3;16,3;19,2;]

Результати для 20 міст(популяція 200):

Finaldistance: 71

Time: 41240мс

Solution:

|16, 12|20, 12|19, 2|16, 3|16, 3|13, 5|11, 6|7, 3|6, 7|1, 13|4, 13|8, 15|5, 18|6, 19|7, 20|13, 17|12, 15|13, 12|11, 10|13, 8|

В програмі використовувалося двох точкове впорядковуючи схрещування , мутація класичне інвертування і турнірна селекція.

Таблиця порівняння кількості міст і популяції

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Кількість міст | 10 | | | 20 | | |
| Популяція | 20 | 100 | 200 | 20 | 100 | 200 |
| Час виконання,c | 2.27 | 11.33 | 22.6 | 4.06 | 19.99 | 41.08 |
| Довжина шляху | 50 | 50 | 50 | 75 | 71 | 71 |
| Довжина шляху в 3 лаб. | 53.6 | 53.6 | 53.6 | 91.38 | 87.22 | 80.76 |

Код програми

**GA.class**

**package** tsp;

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

/\* ГА параметри \*/

**private** **static** **final** **double** *mutationRate* = 0.015;//ймовірність мутації

**private** **static** **final** **int** *truncationSize* = 5;//кількість проходів для турнірної селекції

// Розвивається населенням більше одного покоління

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

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

**int** elitismOffset = 1;

// кросовер населення

// Цикл розміром нового населення і створення осіб поточного населення

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

// Виберір батьків

Tour parent1 = *truncationSelection*(pop);

Tour parent2 = *truncationSelection*(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** **static** Tour crossover(Tour parent1, Tour parent2) {

// створити новий прохід нащадка

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

**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** **static** **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** **static** Tour truncationSelection(Population pop) {

// селекція усканням

Population truncation = **new** Population(*truncationSize*, **false**);

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

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

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

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

}

//взяти найкращий прохід

Tour fittest = truncation.getFittest();

**return** fittest;

}

}

**Class TSP\_GA**

**package**tsp;

**import**java.util.Date;

**publicclass** TSP\_GA {

**publicstaticvoid**main(String[] args) {

**int** [] x = {7, 13, 4, 5, 12, 8, 6, 13, 1, 11, 16, 20, 16, 6, 13, 11, 13, 7, 16, 19};

**int** [] y = {20, 8, 13, 18, 15, 15, 7, 17, 13, 6, 3, 12, 12, 19, 5, 10, 12, 3, 3, 2};

//Створення і додаванняміста

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

TourManager.*addCity*(**new**City(x[i],y[i]));

}

//Ініціалізаціяпопуляції

Populationpop = **new**Population(200, **true**);

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

DatecurrentTimeBefore = **new**Date();

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

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

// Розвиненнянаселеняна 100 поколінь

//

pop = GA.*evolvePopulation*(pop);

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

pop = GA.*evolvePopulation*(pop);

}

DatecurrentTimeAfter = **new**Date();

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

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

**long**time = timeAfter-timeBefore;

//Вивідрезультатів

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

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

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

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

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

}

}

**Висновки:** виконавши лабораторну роботу я вивчив еволюційні оператори схрещування та мутації, що використовуються при розв’язуванні задач комбінаторної оптимізації.Реалізував за допомогою програмної мови Java програмне забезпечення для вирішення задачі комівояжера, використовуючи турнірну селекцію, двох точковий кросинговер і класичне інвертуванняв результаті програма дає результати близькі до оптимальних при кількості міст до 20, при більшій кількості шлях є неоптимальним. Також був виявлений зв'язок що при збільшенні кількості популяції довжина шляху зменшується, але час роботи програми збільшується