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

**Завдання:** Розробити на довільній мові програмування програмне забезпечення для вирішення задачі комівояжера.

В – 9(1)

Використовуючи турнірну селекцію.

Результати виконання програми

При запуску для 10 міст з координатами:

0 14;7 0; 13 12;18 5; 18 9;4 16; 6 11;16 12;13 12;6 19]

Рішення:

|13, 12|13, 12|16, 12|18, 9|18, 5|7, 0|6, 11|0, 14|4, 16|6, 19|

При запуску для 20 міст з координатами) :

14,2;16,0;11,8;5,18;4,7;6,8;7,5;0,19;13,8;6,6;20,15;8,19;17,6;0,5;20,18;0,13;6,10;3,18;12,11;6,18;]

Рішення:

|7, 5|6, 6|4, 7|0, 5|6, 8|6, 10|0, 13|0, 19|3, 18|5, 18|6, 18|8, 19|20, 18|20, 15|12, 11|11, 8|13, 8|17, 6|16, 0|14, 2|

При запуску для 30 міст з координатами:

=[5,17;16,17;15,3;11,3;0,14;14,3;20,4;20,15;3,9;12,4;15,13;8,2;13,0;11,20;12,6;12,1;9,10;16,3;13,20;15,5;20,7;2,12;5,13;19,18;0,8;16,14;14,9;9,12;2,10;2,7;

Рішення:

|15, 13|14, 9|12, 6|14, 3|15, 3|16, 3|20, 4|20, 7|15, 5|12, 4|11, 3|12, 1|13, 0|8, 2|2, 7|0, 8|3, 9|2, 10|0, 14|2, 12|5, 13|9, 10|9, 12|5, 17|11, 20|13, 20|16, 17|19, 18|20, 15|16, 14|

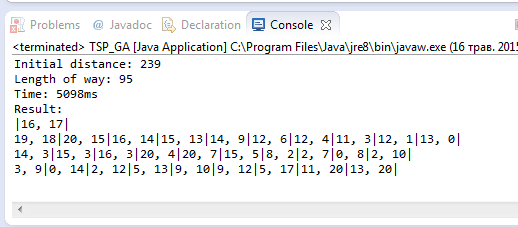


Рис.1. Результат програми при кількості міст 30 і популяції 100.

Таблиця порівняння залежності кількості міст і популяції

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Кількість міст | 10 | | | 20 | | | 30 | | |
| Популяція | 20 | 50 | 100 | 20 | 50 | 100 | 20 | 50 | 100 |
| Час виконання,c | 0.42 | 0.711 | 1.37 | 0.593 | 1.121 | 2.12 | 0.679 | 1.63 | 3.1 |
| Мін. довжинна в Matlab(3 лаб) | 58.41 | 58.41 | 58.41 | 102.34 | 99.27 | 92.1 | 166.27 | 141.96 | 124.25 |
| Мінімальна довжина | 55 | 55 | 55 | 91 | 86 | 82 | 108 | 99 | 94 |

Код програми

**class TSP\_GA**

**package** tsp;

**import** java.util.Date;

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

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

**int** [] x = {5, 16, 15, 11, 0, 14, 20, 20, 3, 12, 15, 8, 13, 11, 12, 12, 9, 16, 13, 15, 20, 2, 5, 19, 0, 16, 14, 9, 2, 2};

**int** [] y = {17, 17, 3, 3, 14, 3, 4, 15, 9, 4, 13, 2, 0, 20, 6, 1, 10, 3, 20, 5, 7, 12, 13, 18, 8, 14, 9, 12, 10, 7};

// Create and add our cities

**for**(**int** i = 0; i < 30; i++){

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

}

// Initialize population

Population pop = **new** Population(50, **true**);

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

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

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

// Evolve population for 10000 generations

pop = GA.*evolvePopulation*(pop);

**for** (**int** i = 0; i < 10000; i++) {

pop = GA.*evolvePopulation*(pop);

}

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

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

**long** time = timeAfter-timeBefore;

// Print final results;

System.*out*.println("Length of way: " + pop.getFittest().getDistance());

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

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

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

}

}

**class GA**

**package** tsp;

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

/\* GA parameters \*/

**private** **static** **final** **double** *mutationRate* = 0.015; //rate of mutation

**private** **static** **final** **int** *tournamentSize* = 5; //size of tournament

// Evolves a population over one generation

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

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

// Keep our best individual if elitism is enabled

**int** elitismOffset = 1;

newPopulation.saveTour(0, pop.getFittest());

// Crossover population

// Loop over the new population's size and create individuals from

// Current population

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

// Select parents

Tour parent1 = *tournamentSelection*(pop);

Tour parent2 = *tournamentSelection*(pop);

// Crossover parents

Tour child = *crossover*(parent1, parent2);

// Add child to new population

newPopulation.saveTour(i, child);

}

// Mutate the new population a bit to add some new genetic material

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

*mutate*(newPopulation.getTour(i));

}

**return** newPopulation;

}

// Applies crossover to a set of parents and creates offspring

//1point crossover

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

// Create new child tour

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

// Get start and end sub tour positions for parent1's tour

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

// Loop and add the sub tour from parent1 to our child

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

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

}

// Loop through parent2's city tour

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

// If child doesn't have the city add it

**if** (!child.containsCity(parent2.getCity(i))) {

// Loop to find a spare position in the child's tour

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

// Spare position found, add city

**if** (child.getCity(ii) == **null**) {

child.setCity(ii, parent2.getCity(i));

**break**;

}

}

}

}

**return** child;

}

// Mutate a tour using swap mutation

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

// Loop through tour cities

**for**(**int** tourPos1=0; tourPos1 < tour.tourSize(); tourPos1++){

// Apply mutation rate

**if**(Math.*random*() < *mutationRate*){

// Get a second random position in the tour

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

// Get the cities at target position in tour

City city1 = tour.getCity(tourPos1);

City city2 = tour.getCity(tourPos2);

// Swap them around

tour.setCity(tourPos2, city1);

tour.setCity(tourPos1, city2);

}

}

}

// Selects candidate tour for crossover

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

// Create a tournament population

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

// For each place in the tournament get a random candidate tour and

// add it

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

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

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

}

// Get the fittest tour

Tour fittest = tournament.getFittest();

**return** fittest;

}

}

**Висновки:** виконавши лабораторну роботу я вивчив еволюційні оператори, що використовуються при розв’язуванні задач комбінаторної оптимізації. Реалізував за допомогою мови програмування Java програмне забезпечення для вирішення задачі комівояжера з одноточковим схрещуванням і мутацією обміну. В результаті програма коректно працює для кількість міст до 10, з більшою кількістю міст шлях комівояжера не оптимальний, але програма показує набагато кращі результати у порівнянні з реалізацією за допомогою пакету Matlab в 3 лабораторній роботі.