|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| СПКс-11 | 5 | **Програмування ГА для задачі Комівояжера (TSP)** |  |  |
| Киценюк М.Л. | |
| № залікової: 1508503 | |
| 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.

**Індивідуальне завдання**

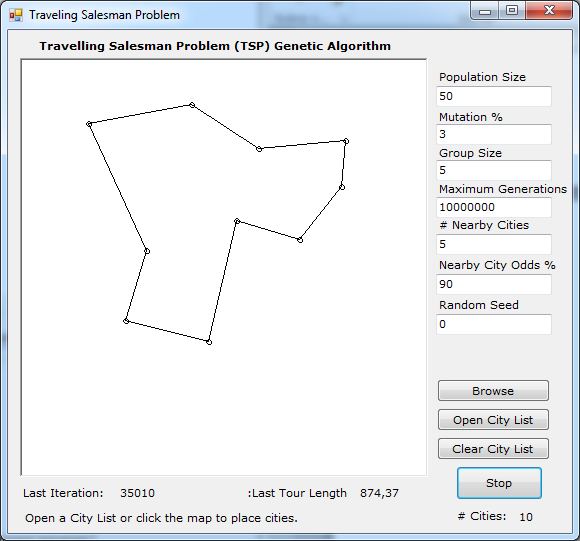
У вас є безліч міст (представлені у вигляді точок на площині з X і Y координати). Мета полягає в тому, щоб знайти найкоротший маршрут, який відвідує кожне місто рівно один раз, повертаючись в кінці своєї відправної точки.

Дано від 10 до 50 точок. Мова програмування довільна.

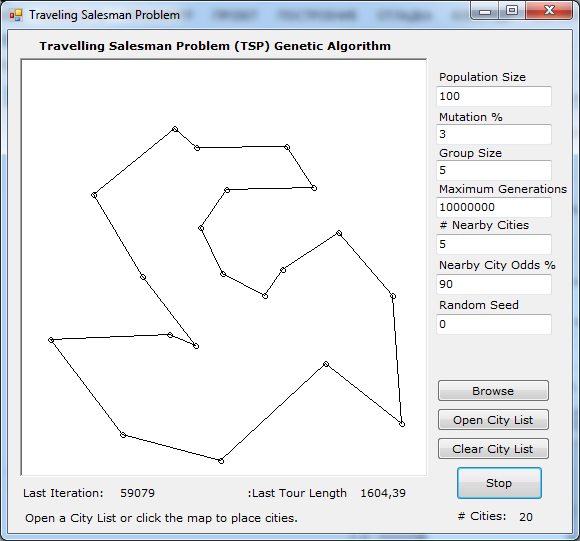
4. Пропорційний відбір

**РЕЗУЛЬТАТИ ВИКОНАННЯ РОБОТИ**

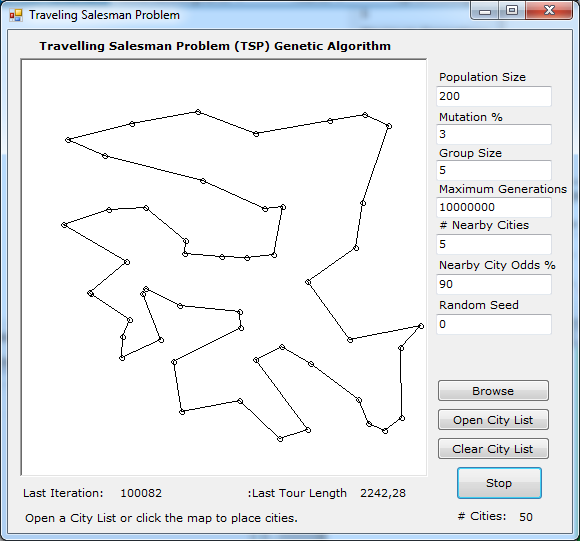
У розробленій програмі користувач має змогу самостійно вказувати такі параметри як: кількість міст, кількість поколінь, розмір популяції та відсоток мутації та інше. Використання параметру “сусідні міста” дозволяє покращити результати алгоритму, оскільки зв’язок між містами приорітетно встановлюється між найближчими містами.



*Рис.1.* Результати роботи алгоритму для 10 міст



*Рис.2.* Результати роботи алгоритму для 20 міст



*Рис.3.* Результати роботи алгоритму для 50 міст

Тестувати ефективність роботи алгоритму проведено на основі трьох видів задачі комівояжера – для 10, 20 та 50 міст (тобто кількості змінних).Ефективність роботи ГА для задачі комівояжера прийнято оцінювати на основі довжини шляху. Чим значення цього параметру менше, тим краще.

Для порівняння отриманих результатів складемо таблицю з характеристиками роботи генетичного алгоритму для різних видів задачі комівояжера.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Кількість міст** | | |
| 10 | 20 | 50 |
| **Характеристики алгортиму** |  | | |
| Розмір популяції: | 50 | 100 | 200 |
| Кільсть поколінь: | 35010 | 59079 | 100082 |
| Довжина маршруту: | 874.37 | 1604.39 | 2242.28 |

**ЛІСТИНГ ПРОГРАМИ**

Лістинг файлу “Tour.cs”

using System;

using System.Collections.Generic;

using System.Text;

namespace Tsp

{

// клас представляє один примірник туру по всіх містах

public class Tour : List<Link>

{

m\_fitnessTable = new ArrayList();

m\_thisGeneration = new ArrayList(m\_generationSize);

m\_nextGeneration = new ArrayList(m\_generationSize);

public Tour(int capacity) : base(capacity) {

resetTour(capacity);

}

private double fitness;

public double Fitness {

set { fitness = value; }

get { return fitness; }

}

//створює тур з правильним кількістю міст і створює початкові підключення всіх -1

private void resetTour(int numberOfCities) {

this.Clear();

Link link;

for (int i = 0; i < numberOfCities; i++) {

link = new Link();

link.Connection1 = -1;

link.Connection2 = -1;

this.Add(link);

}

}

// визначення загальної довжини окремого туру

public void DetermineFitness(Cities cities) {

Fitness = 0;

int lastCity = 0;

int nextCity = this[0].Connection1;

foreach (Link link in this) {

Fitness += cities[lastCity].Distances[nextCity];

// в списку, якщо міто проруч чи ні,то [0] або [1]

if (lastCity != this[nextCity].Connection1) {

lastCity = nextCity;

nextCity = this[nextCity].Connection1;

}

else {

lastCity = nextCity;

nextCity = this[nextCity].Connection2;

}

}

}

// створення зв’язку між 2 мітсами

private static void joinCities(Tour tour, int[] cityUsage, int city1, int city2) {

// Determine if the [0] or [1] link is available in the tour to make this link.

if (tour[city1].Connection1 == -1) {

tour[city1].Connection1 = city2;

}

else {

tour[city1].Connection2 = city2;

}

if (tour[city2].Connection1 == -1) {

tour[city2].Connection1 = city1;

}

else {

tour[city2].Connection2 = city1;

}

cityUsage[city1]++;

cityUsage[city2]++;

}

// пошук міста в батьківському турі, який може бути занесений в тур дитини

//якщо обидва посилання в батьків не є допустимими посилання на тур з дітьми, повернути -1 .

private static int findNextCity(Tour parent, Tour child, Cities cityList, int[] cityUsage, int city) {

if (testConnectionValid(child, cityList, cityUsage, city, parent[city].Connection1)) {

return parent[city].Connection1;

}

else if (testConnectionValid(child, cityList, cityUsage, city, parent[city].Connection2)) {

return parent[city].Connection2;

}

return -1;

}

private static bool testConnectionValid(Tour tour, Cities cityList, int[] cityUsage, int city1, int city2) {

// перевірка чи мають міста сусідів

if ((city1 == city2) || (cityUsage[city1] == 2) || (cityUsage[city2] == 2)) {

return false;

}

// перевірка чи збережений зв'язок між 2 містами

if ((cityUsage[city1] == 0) || (cityUsage[city2] == 0)) {

return true;

}

// перевірка чи всі міста зв’язані

for (int direction = 0; direction < 2; direction++) {

int lastCity = city1;

int currentCity;

if (direction == 0) {

currentCity = tour[city1].Connection1

}

else {

currentCity = tour[city1].Connection2;

}

int tourLength = 0;

while ((currentCity != -1) && (currentCity != city2) && (tourLength < cityList.Count - 2)) {

tourLength++;

// якщо місто в списку то [0] чи [1]

if (lastCity != tour[currentCity].Connection1) {

lastCity = currentCity;

currentCity = tour[currentCity].Connection1;

}

else {

lastCity = currentCity;

currentCity = tour[currentCity].Connection2;

}

}

// якщо міста пов'язані, але зв'язок не проходить через кожне місто в списку, то з’єднати їх

if (tourLength >= cityList.Count - 2) {

return true;

}

if (currentCity == city2) {

return false;

}

}

return true;

}

**// пропорційний відбір**

private static Tour ProportionalSelection()

{

double randomFitness = m\_random.NextDouble() \* m\_totalFitness;

int idx = -1;

int mid;

int first = 0;

int last = m\_populationSize -1;

mid = (last - first)/2;

// ArrayList's BinarySearch is for exact values only

// so do this by hand.

while (idx == -1 && first <= last)

{

if (randomFitness < (double)m\_fitnessTable[mid])

{

last = mid;

}

else if (randomFitness > (double)m\_fitnessTable[mid])

{

first = mid;

}

mid = (first + last)/2;

// lies between i and i+1

if ((last - first) == 1)

idx = last;

}

return idx;

}

public static Tour Crossover(Tour parent1, Tour parent2, Cities cityList, Random rand) {

Tour child = new Tour(cityList.Count); // the new tour we are making

int[] cityUsage = new int[cityList.Count]; // how many links 0-2 that connect to this city

int city; // for loop variable

int nextCity; // the other city in this link

// Take all links that both parents agree on and put them in the child

for (city = 0; city < cityList.Count; city++) {

if (cityUsage[city] < 2) {

if (parent1[city].Connection1 == parent2[city].Connection1) {

nextCity = parent1[city].Connection1;

if (testConnectionValid(child, cityList, cityUsage, city, nextCity)) {

joinCities(child, cityUsage, city, nextCity);

}

}

if (parent1[city].Connection2 == parent2[city].Connection2) {

nextCity = parent1[city].Connection2;

if (testConnectionValid(child, cityList, cityUsage, city, nextCity)) {

joinCities(child, cityUsage, city, nextCity);

}

}

if (parent1[city].Connection1 == parent2[city].Connection2) {

nextCity = parent1[city].Connection1;

if (testConnectionValid(child, cityList, cityUsage, city, nextCity)) {

joinCities(child, cityUsage, city, nextCity);

}

}

if (parent1[city].Connection2 == parent2[city].Connection1) {

nextCity = parent1[city].Connection2;

if (testConnectionValid(child, cityList, cityUsage, city, nextCity)) {

joinCities(child, cityUsage, city, nextCity);

}

}

}

}

// The parents don't agree on whats left, so we will alternate between using

// links from parent 1 and then parent 2.

for (city = 0; city < cityList.Count; city++) {

if (cityUsage[city] < 2) {

// we prefer to use parent 1 on odd cities

if (city % 2 == 1) {

nextCity = findNextCity(parent1, child, cityList, cityUsage, city);

// but if thats not possible we still go with parent 2

if (nextCity == -1) {

nextCity = findNextCity(parent2, child, cityList, cityUsage, city); ;

}

}

else {

// use parent 2 instead

nextCity = findNextCity(parent2, child, cityList, cityUsage, city);

if (nextCity == -1) {

nextCity = findNextCity(parent1, child, cityList, cityUsage, city);

}

}

if (nextCity != -1) {

joinCities(child, cityUsage, city, nextCity);

if (cityUsage[city] == 1) {

if (city % 2 != 1) {

nextCity = findNextCity(parent1, child, cityList, cityUsage, city);

if (nextCity == -1) {

nextCity = findNextCity(parent2, child, cityList, cityUsage, city);

}

}

else {

nextCity = findNextCity(parent2, child, cityList, cityUsage, city);

if (nextCity == -1) {

nextCity = findNextCity(parent1, child, cityList, cityUsage, city);

}

}

if (nextCity != -1) {

joinCities(child, cityUsage, city, nextCity);

}

}

}

}

}

// Parent's links would cause multiple disconnected loops.

for (city = 0; city < cityList.Count; city++) {

while (cityUsage[city] < 2) {

do {

nextCity = rand.Next(cityList.Count);

} while (!testConnectionValid(child, cityList, cityUsage, city, nextCity));

joinCities(child, cityUsage, city, nextCity);

}

}

return child;

}

// Randomly change one of the links in this tour.

public void Mutate(Random rand) {

int cityNumber = rand.Next(this.Count);

Link link = this[cityNumber];

int tmpCityNumber;

// Find which 2 cities connect to cityNumber, and then connect them directly

if (this[link.Connection1].Connection1 == cityNumber) {

if (this[link.Connection2].Connection1 == cityNumber) {

tmpCityNumber = link.Connection2;

this[link.Connection2].Connection1 =link.Connection1;

this[link.Connection1].Connection1 = tmpCityNumber;

}

else {

tmpCityNumber = link.Connection2;

this[link.Connection2].Connection2 = link.Connection1;

this[link.Connection1].Connection1 = tmpCityNumber;

}

}

else {

if (this[link.Connection2].Connection1 == cityNumber) {

tmpCityNumber = link.Connection2;

this[link.Connection2].Connection1 = link.Connection1;

this[link.Connection1].Connection2 = tmpCityNumber;

}

else {

tmpCityNumber = link.Connection2;

this[link.Connection2].Connection2 = link.Connection1;

this[link.Connection1].Connection2 = tmpCityNumber;

}

}

int replaceCityNumber = -1;

do {

replaceCityNumber = rand.Next(this.Count);

}

while (replaceCityNumber == cityNumber);

Link replaceLink = this[replaceCityNumber];

tmpCityNumber = replaceLink.Connection2;

link.Connection2 = replaceLink.Connection2;

link.Connection1 = replaceCityNumber;

replaceLink.Connection2 = cityNumber;

if (this[tmpCityNumber].Connection1 == replaceCityNumber) {

this[tmpCityNumber].Connection1 = cityNumber;

}

else {

this[tmpCityNumber].Connection2 = cityNumber;

}

}}}

**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 C#, що дозволяє розв’язувати різні види задач комівояжера з застосування генетичних алгоритмів.