Прізвище: Дробко

Ім’я: Володимир

Група: КНС-13

Дата прийняття роботи

у системі Git: ХХ.05.2017

Дисципліна: Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні

Перевірив: Кривий Р.З.

**Звіт до лабораторної роботи № 3**

**«Комбінаторна оптимізація за допомогою**

**еволюційних методів»**

**МЕТА РОБОТИ**

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

**КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ**

При використанні методів еволюційного пошуку для розв’язку задач комбінаторної оптимізації, як правило, застосовуються негомологічні числові хромосоми, тобто такі хромосоми, гени яких можуть приймати значення в заданому інтервалі. При цьому інтервал однаковий для всіх генів, але в хромосомі не може бути двох генів з однаковим значенням. Комбінаторні задачі оперують із дискретними структурами або розміщенням об'єктів, незначні зміни яких часто викликають стрибкоподібну зміну показників якості (фітнес-функції).Традиційні оператори еволюційні оператори, що генерують нових нащадків, не можуть бути застосовані при використанні негомологічних хромосом, оскільки внаслідок виконання таких операторів генеруються нащадки, що містять однакові гени і тому не можуть бути інтерпретовані при розв’язку комбінаторної задачі. Тому для розв’язку задач комбінаторної оптимізації були розроблені спеціальні генетичні оператори, що не створюють неприпустимих рішень.

Циклове схрещуванні (Cycle crossover) одержує нове рішення С1 шляхом композиції вихідної підстановки Р2 і деякої підстановки ( ) s s sk 0 , 1 ,..., , що представляє собою цикл довжини k + 1, тобто ( ) C P s s sk 1 = 2 ο 0 , 1 ,..., , при цьому ( ( )); 1 0 1 s1 P2 P s − = ( ( )); 1 1 1 s2 P2 P s − = ………………… ( ( )). 1 1 1 2 − − sk = P P sk Тут s0 – випадково обране число від 1 до L – 1 і ( ( )) 1 0 1 sk 1 = P2 P sk = s − + . Рішення С2 виходить аналогічним шляхом за допомогою заміни Р2 на Р1 у попередніх виразах.

**Одноточечна мутація обміну**

Одноточечна мутація обміну. У даному операторі, на відміну від попереднього, обмінюються місцями тільки сусідні гени, і точка мутації вибирається між двома генами. Крок 1. Створити хромосому нащадка як копію батьківської хромосоми H = {h1,h2,..., hL}. 12 Крок 2. Вибрати точку мутації y випадковим чином із множини Y = {1, 2, .... L- 1}. Крок 3. Сформувати нову хромосому H шляхом обміну елементів, розташованих на позиціях y і y+1. Таким чином, після застосування одноточечної мутації обміну одержуємо хромосому H': H′ = {h1 ,h2 ,...,hy+1 ,hy ,...,hL} .

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

Розробити за допомогою пакету Matlab програмне забезпечення для вирішення задачі комівояжера. Параметри еволюційного методу обрати з таблиці 1 відповідно до варіанту.

*Таблиця 1*

|  |  |  |
| --- | --- | --- |
| № варіанту | Еволюційні оператори | |
| Схрещування | Мутація |
| 7 | циклове | Одноточечна мутація обміну |

**Код головного файлу програми:**

function varargout = lab3(varargin)

% Ініціалізація початкових змінних

defaultConfig.xy = 50\*rand(50,2);

defaultConfig.matrixd = [];

defaultConfig.legSize = 30;

defaultConfig.quantIter = 1e4;

defaultConfig.getProg = true;

defaultConfig.getResult = true;

defaultConfig.getWaitbar = false;

% Інтерпретація вхідних данних

if ~nargin

userConfig = struct();

elseif isstruct(varargin{1})

userConfig = varargin{1};

end

configStruct = get\_config(defaultConfig,userConfig);

% Конфігурація програми

xy = configStruct.xy;

matrixd = configStruct.matrixd;

legSize = configStruct.legSize;

quantIter = configStruct.quantIter;

getProg = configStruct.getProg;

getResult = configStruct.getResult;

getWaitbar = configStruct.getWaitbar;

if isempty(matrixd)

nPoints = size(xy,1);

a = meshgrid(1:nPoints);

matrixd = reshape(sqrt(sum((xy(a,:)-xy(a',:)).^2,2)),nPoints,nPoints);

end

% Перевірка вхідних даних

[N,dims] = size(xy);

[nr,nc] = size(matrixd);

n = N;

% Перевірка вх даних

legSize = 4\*ceil(legSize/4);

quantIter = max(1,round(real(quantIter(1))));

getProg = logical(getProg(1));

getResult = logical(getResult(1));

getWaitbar = logical(getWaitbar(1));

% Створення популяції

pop = zeros(legSize,n);

pop(1,:) = (1:n);

for k = 2:legSize

pop(k,:) = randperm(n);

end

% Запусе ГА

globalMin = Inf;

totalDist = zeros(1,legSize);

distHistory = zeros(1,quantIter);

tmpPop = zeros(4,n);

newPop = zeros(legSize,n);

if getProg

figure('Name','TSP\_GA | Current Best Solution','Numbertitle','off');

hAx = gca;

end

if getWaitbar

end

for iter = 1:quantIter

% Обчислення кожного члена

for p = 1:legSize

d = matrixd(pop(p,n),pop(p,1)); % Closed Path

for k = 2:n

d = d + matrixd(pop(p,k-1),pop(p,k));

end

totalDist(p) = d;

end

% Знаходження найкращого шляху

[minDist,index] = min(totalDist);

distHistory(iter) = minDist;

if minDist < globalMin

globalMin = minDist;

optRoute = pop(index,:);

if getProg

% Графік найкращого шляху

rte = optRoute([1:n 1]);

if dims > 2, plot3(hAx,xy(rte,1),xy(rte,2),xy(rte,3),'r.-');

else plot(hAx,xy(rte,1),xy(rte,2),'r.-'); end

title(hAx,sprintf('Full Distance = %1.4f, Iteration = %d',minDist,iter));

drawnow;

end

end

% Оператори генетичного алгоритму

randomOrder = randperm(legSize);

for p = 4:4:legSize

rtes = pop(randomOrder(p-3:p),:);

dists = totalDist(randomOrder(p-3:p));

[ignore,idx] = min(dists); bestOf4Route = rtes(idx,:);

routeInsertionPoints = sort(ceil(n\*rand(1,2)));

I = routeInsertionPoints(1);

J = routeInsertionPoints(2);

for k = 1:4 % Мутація

tmpPop(k,:) = bestOf4Route;

switch k

case 2

tmpPop(k,I:J) = tmpPop(k,J:-1:I);

case 3

tmpPop(k,[I J]) = tmpPop(k,[J I]);

case 4

tmpPop(k,I:J) = tmpPop(k,[I+1:J I]);

otherwise

end

end

newPop(p-3:p,:) = tmpPop;

end

pop = newPop;

% Оновлення

if getWaitbar && ~mod(iter,ceil(quantIter/325))

waitbar(iter/quantIter,hWait);

end

end

if getWaitbar

close(hWait);

end

% Повернення вих даних

if nargout

resultStruct = struct( ...

'xy', xy, ...

'matrixd', matrixd, ...

'legSize', legSize, ...

'quantIter', quantIter, ...

'getProg', getProg, ...

'getResult', getResult, ...

'getWaitbar', getWaitbar, ...

'optRoute', optRoute, ...

'minDist', minDist);

varnewout = {resultStruct};

end

end

% Зміна початкової конфігурації вхідних параметрів

function config = get\_config(defaultConfig,userConfig)

% Ініціалізація вхідної конфігурації

config = defaultConfig;

defaultFields = fieldnames(defaultConfig);

userFields = fieldnames(userConfig);

nUserFields = length(userFields);

% Зміна усієї стандартної конфігурації разом з даними користувача

for i = 1:nUserFields

userField = userFields{i};

isField = strcmpi(defaultFields,userField);

if nnz(isField) == 1

thisField = defaultFields{isField};

config.(thisField) = userConfig.(userField);

end

end

end

**Результат виконання лабораторного завдання.**

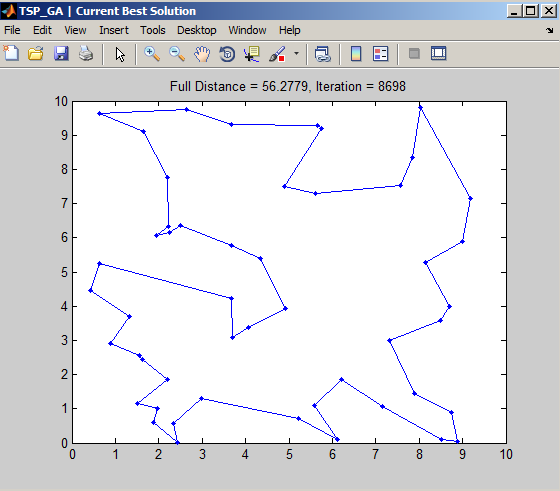


Рис 1. Розв’язок при параметрах: кількість генерацій 60, кількість міст 20

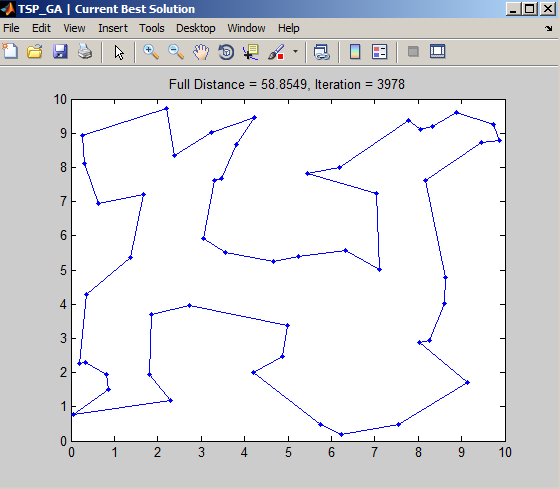


Рис 2. Розв’язок при параметрах: кількість генерацій 120, кількість міст 30

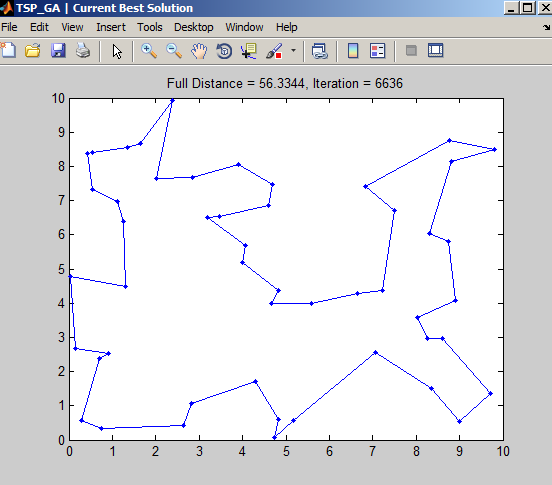


Рис 3. Розв’язок при параметрах: кількість генерацій 240, кількість міст 40

**Висновки.**

Виконуючи лабораторну роботу я ознайомився з основними теоретичними відомостями, вивчив еволюційні оператори схрещування та мутації, що використовуються при розв’язуванні задач комбінаторної оптимізації. А також реалізував програму, використовуючи Matlab, для реалізації розв’язку задачі комівояжера. При великій кількості міст генетичний алгоритм потребує великої кількості поколінь для знаходження оптимального шляху, і навіть тоді результат не є прийнятним.