|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| КНС-13 | 3 | Комбінаторна оптимізація за допомогою еволюційних методів |  |  |
| Назар Богдан | |
|  | |
| Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні | | Викладач: | |
| Кривий Р.З. | |

**1.Мета**

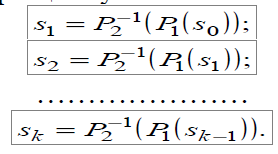
Ознайомитися з основними теоретичними відомостями, вивчити еволюційні оператори схрещування та мутації, що використовуються при розв’язуванні задач комбінаторної оптимізації.

1. **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 (фітнесс-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 схрещування**

Циклове схрещуванні (Cycle crossover) одержує нове рішення *С*1 шляхом композиції вихідної підстановки *Р*2 і деякої підстановки  що представляє собою цикл довжини *k* + 1, тобто , при цьому



Тут *s*0 – випадково обране число від 1 до *L* – 1 і 

. Рішення *С*2 виходить аналогічним шляхом за допомогою заміни *Р*2 на *Р*1 у

попередніх виразах*.*

**Мутація золотого перетину**.

У даному операторі вибір точки мутації здійснюється на основі правила “золотого перетину”, тобто точка мутації хромосом довжини *L* визначається за формулою: D = Ціле(τ·*L*), де  В результаті застосування оператора мутації

золотого перетину хромосома *H* = {*h*1,*h*2,…,*h*D,*h*D+1,...,*hL*} перетворюється у

хромосому *H* = {*h*1,*h*2,…,*h*D+1,*h*D,...,*hL*}.

**Індивідуальне завдання**

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

Розробити за допомогою пакету Matlab програмне забезпечення для вирішення задачі комівояжера.

**Варіант №13(3)**

|  |  |  |
| --- | --- | --- |
| **№** | **Схрещування** | **Мутація** |
| 3 | циклове | Мутація золотого перетину |

**Код програми на мові Matlab :**

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};

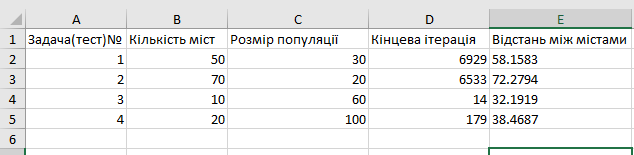
config.(thisField) = userConfig.(userField);

end

end

end

**Результати виконання:**



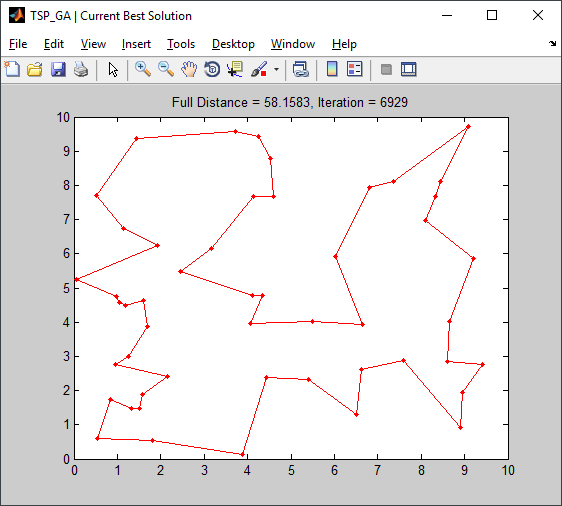


Рис. 1 Результат роботи програми для Тест № 1

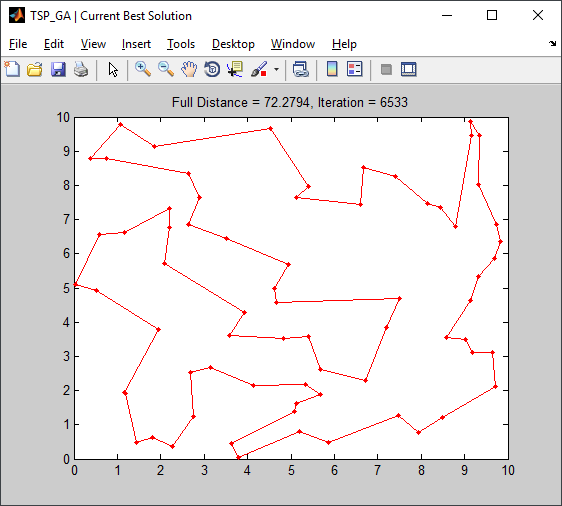


Рис. 2 Результат роботи програми для Тест № 2

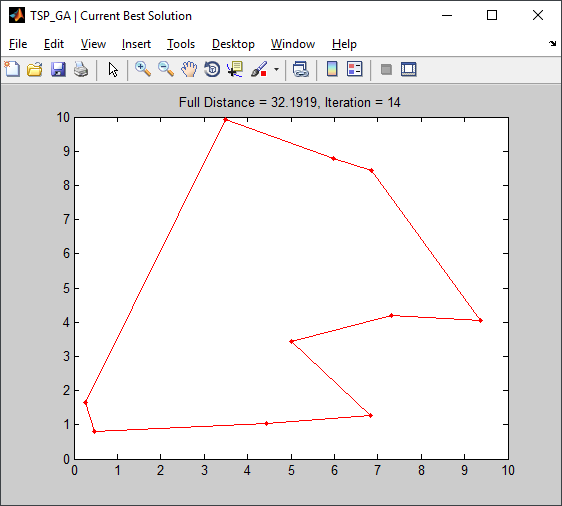


Рис. 3 Результат роботи програми для Тест № 3

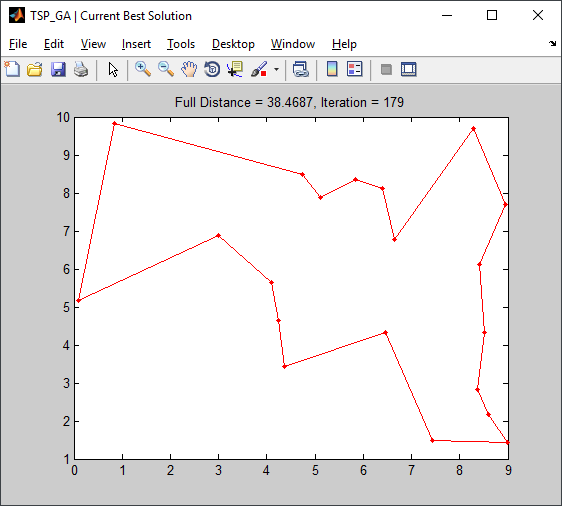


Рис. 4 Результат роботи програми для Тест № 4

**4. Висновок**

Виконуючи лабораторну роботу я ознайомився з основними еволюційними операторами схрещування та мутації, що використовуються при розв’язуванні задач комбінаторної оптимізації. Розробив за допомогою пакету Matlab програмне забезпечення для вирішення задачі комівояжера.