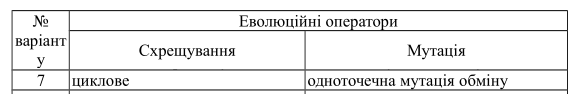
**Мета роботи:**

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

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

**Варіант -6**

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



**Порядок виконання роботи**

1. Лістинг програми.

***mainWindow.m***

function varargout = mainWindow(varargin)

% Begin initialization code - DO NOT EDIT

gui\_Singleton = 1;

gui\_State = struct('gui\_Name', mfilename, ...

'gui\_Singleton', gui\_Singleton, ...

'gui\_OpeningFcn', @mainWindow\_OpeningFcn, ...

'gui\_OutputFcn', @mainWindow\_OutputFcn, ...

'gui\_LayoutFcn', [] , ...

'gui\_Callback', []);

if nargin && ischar(varargin{1})

gui\_State.gui\_Callback = str2func(varargin{1});

end

if nargout

[varargout{1:nargout}] = gui\_mainfcn(gui\_State, varargin{:});

else

gui\_mainfcn(gui\_State, varargin{:});

end

% End initialization code - DO NOT EDIT

% --- Executes just before mainWindow is made visible.

function mainWindow\_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;

guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.

function varargout = mainWindow\_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

% --- Executes on button press in btnSolve.

function btnSolve\_Callback(hObject, eventdata, handles)

borderString = get(handles.listboxBorder, 'string');

v = get(handles.listboxBorder, 'value'); %get current selection value

borderString = borderString{v}; %choose corresponding cell

load(borderString,'x','y','xx','yy');

%%

% Generating random locations of cities inside the border.

% We use the INPOLYGON function to make sure that all

% the cities are inside or very close to the boundary.

citiesString = get(handles.editCitiesAmount,'String');

cities = STR2NUM(citiesString);

locations = zeros(cities,2);

n = 1;

while (n <= cities)

xp = rand\*1.5;

yp = rand;

if inpolygon(xp,yp,xx,yy)

locations(n,1) = xp;

locations(n,2) = yp;

n = n+1;

end

end

%%

% Given the list of city locations,

% we can calculate the distance matrix for all the cities.

distances = zeros(cities);

for count1=1:cities,

for count2=1:count1,

x1 = locations(count1,1);

y1 = locations(count1,2);

x2 = locations(count2,1);

y2 = locations(count2,2);

distances(count1,count2)=sqrt((x1-x2)^2+(y1-y2)^2);

distances(count2,count1)=distances(count1,count2);

end;

end;

%%

% The custom crossover function takes a cell array, the population, and

% returns a cell array, the children that result from the crossover.

type CXcrossover.m

%%

% The custom mutation function takes an individual, which is an ordered set

% of cities, and returns a mutated ordered set.

type mutateSalesman.m

%%

% Fitness function for the traveling salesman problem. The

% fitness of an individual is the total distance traveled for an ordered

% set of cities. The fitness function also needs the distance matrix to

% calculate the total distance.

type salesmanFitness.m

%%

% GA call fitness function with just arguments 'x' and 'distances'.

% We can use an anonymous function to capture the values of the additional argument, the distances matrix.

% We create a function handle 'FitnessFcn' to an anonymous function

% that takes one input 'x', but calls 'salesmanFitness' with x,and distances.

% The variable, distances has a value when the function handle'FitnessFcn'

% is created, so these values are captured by the anonymous function.

FitnessFcn = @(x) salesmanFitness(x,distances);

%%

% A custom plot function to plot the location of the cities and the current best route.

type plotSalesman.m

%%

% Once again we will use an anonymous function to create a function handle

% to an anonymous function which calls 'plotSalesman' with the

% additional argument 'locations'.

my\_plot = @(options,state,flag) plotSalesman(options, ...

state,flag,locations,borderString);

%% Genetic Algorithm Options Setup

% This is an options structure to indicate a custom data type

% and the population range.

options = gaoptimset('PopulationType', 'custom','PopInitRange', ...

[1;cities]);

%%

% Choose the custom creation, crossover, mutation, and plot functions that

% we have created, as well as setting some stopping conditions.

generationsString = get(handles.editGenerations,'String');

populationString = get(handles.editPopulation,'String');

generationsNum = STR2NUM(generationsString);

populationNum = STR2NUM(populationString);

cities = STR2NUM(citiesString);

options = gaoptimset(options,'CreationFcn',@createSalesman, ...

'CrossoverFcn',@CXcrossover, ...

'MutationFcn',@mutateSalesman, ...

'PlotFcn', my\_plot, ...

'Generations',generationsNum,'PopulationSize',populationNum, ...

'StallGenLimit',generationsNum,'Vectorized','on');

%%

% Finally, we call the genetic algorithm with our problem information.

numberOfVariables = cities;

[x,fval,reason,output] = ga(FitnessFcn,numberOfVariables,options)

%%END of main function

function editCitiesAmount\_Callback(hObject, eventdata, handles)

function editCitiesAmount\_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function editPopulation\_Callback(hObject, eventdata, handles)

function editPopulation\_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function editGenerations\_Callback(hObject, eventdata, handles)

function editGenerations\_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function edit4\_Callback(hObject, eventdata, handles)

function edit4\_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function listboxBorder\_Callback(hObject, eventdata, handles)

function listboxBorder\_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

%%END of file ‘mainWindow.m’

***createSalesman.m***

function pop = createSalesman(NVARS,FitnessFcn,options)

%CREATE\_PERMUTATIONS Creates a population of permutations.

totalPopulationSize = sum(options.PopulationSize);

n = NVARS;

pop = cell(totalPopulationSize,1);

for i = 1:totalPopulationSize

pop{i} = randperm(n);

end

%% END of file ‘createSalesman.m’

***salesmanFitness.m***

function scores = salesmanFitness(x,distances)

%TRAVELING\_SALESMAN\_FITNESS Custom fitness function for TSP.

scores = zeros(size(x,1),1);

for j = 1:size(x,1)

p = x{j};

f = distances(p(end),p(1));

for i = 2:length(p)

f = f + distances(p(i-1),p(i));

end

scores(j) = f;

end

%% END of file ‘salesmanFitness.m’

***plotSalesman.m***

function state = plotSalesman(options,state,flag,locations,borderString)

% TRAVELING\_SALESMAN\_PLOT Custom plot function for traveling salesman.

persistent x y xx yy

if strcmpi(flag,'init')

load(borderString,'x','y','xx','yy');

end

plot(x,y,'Color','red');

axis([-0.1 1.5 -0.2 1.2]);

hold on;

[unused,i] = min(state.Score);

genotype = state.Population{i};

***classic\_mutate.m***

function mutationChildren = classic\_mutate(parents ,options,NVARS, ...

FitnessFcn, state, thisScore,thisPopulation,mutationRate)

mutationChildren = cell(length(parents),1);

numberOfGenes=length(thisPopulation{parents(1)});

for i=1:length(parents)

parent = thisPopulation{parents(i)};

XorPoint=mod(ceil(rand(1)\*10),numberOfGenes );

if XorPoint==0

XorPoint=XorPoint+1;

end

child = parent;

child(XorPoint) = parent(XorPoint+1);

child(XorPoint+1) = parent(XorPoint);

mutationChildren{i} = child;

end

***OXcrossover.m***

function xoverKids = OXcrossover(parents,options,NVARS, ...

FitnessFcn,thisScore,thisPopulation)

nKids = length(parents)/2;

xoverKids = cell(nKids,1);

index = 1;

kidsIterator=1;

numberOfGenes=length(thisPopulation{parents(index)});

while kidsIterator<nKids

parent1 = thisPopulation{parents(index)};

index = index + 1;

parent2 = thisPopulation{parents(index)};

index = index + 1;

XorPoint=mod(ceil(rand(1)\*10),numberOfGenes );

if XorPoint==0

XorPoint=XorPoint+1;

end

child1=parent2;

l=XorPoint+1;

for firstPartIterator=1:XorPoint

if ismember(parent1(firstPartIterator), parent2((XorPoint+1):numberOfGenes))==0

child1(firstPartIterator)=parent1(firstPartIterator);

else

for secondPartIterator=l:numberOfGenes

if ismember(parent1(secondPartIterator), parent2((XorPoint+1):numberOfGenes))==0

child1(firstPartIterator)=parent1(secondPartIterator); l=l+1; break;

else

l=l+1;

end

end

end

end

xoverKids{kidsIterator} = child1;

kidsIterator=kidsIterator+1;

child2=parent1;

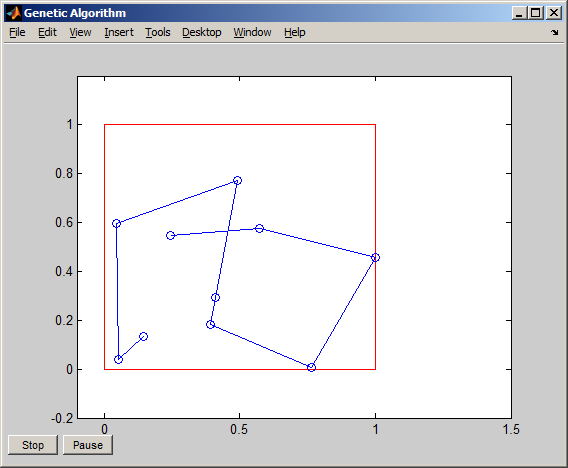
xoverKids{kidsIterator} = child2;

kidsIterator=kidsIterator+1;

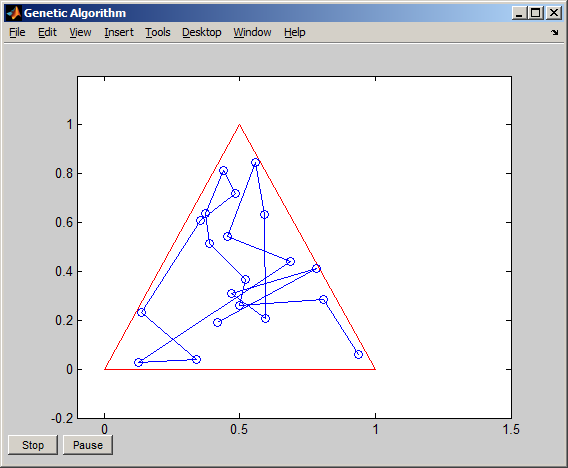
end

1. Результат виконання завдання

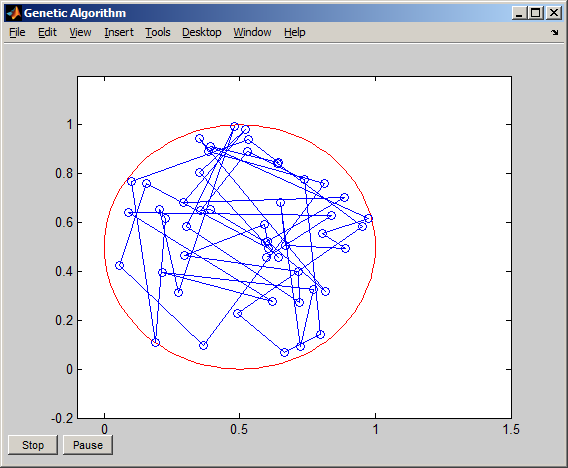
Для зручного використання написаної програми розроблено графічний інтерфейс, що дає змогу самостійно вказувати такі параметри як: кількість міст, кількість поколінь, розмір популяції та обмежуючу площу. Зокрема, границі можуть бути задані квадратом (рис.1), трикутником (рис.2) та колом (рис.3).



*Рис.1. Рішення задачі комівояжера для 10 міст.*



*Рис.2. Рішення задачі комівояжера для 20 міст.*



*Рис.3. Рішення задачі комівояжера для 50 міст.*

Результат виконання програми:

Optimization terminated: maximum number of generations exceeded.

x = [1x50 double]

fval = 18.3029

reason = 0

output =

problemtype: 'unconstrained'

rngstate: [1x1 struct]

generations: 100

funccount: 2020

message: [1x64 char]

*Таблиця 1.*

**Таблиця порівняння кількості міст і популяції**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Кількість міст | 10 | | | 20 | | | 50 | | |
| Покоління | 100 | 200 | 300 | 100 | 200 | 300 | 100 | 200 | 300 |
| Популяція | 20 | 30 | 50 | 20 | 30 | 50 | 20 | 30 | 50 |
| Довжина шляху | 3.05 | 2.15 | 2.65 | 7.46 | 8.02 | 5.87 | 18.7 | 18.69 | 17.21 |
| Час виконання,c | 0.45 | 1.24 | 3.25 | 0.5 | 1.5 | 4.17 | 1.24 | 3.47 | 7.39 |

Тестувати ефективність роботи алгоритму було 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.

**Висновок:** під час виконання лабораторної роботи, я ознайомився з основними принципами застосування еволюційних методів до задач комбінаторної оптимізації. Згідно з лабораторним завданням я написав програму за допомогою середовища Matlab, що дозволяє розв’язувати різні види задач комівояжера з застосування генетичних алгоритмів.