*дата :*

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

**Мета:** 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.

**Теоретичні відомості**

* *Жадібний оператор схрещування*

Жадібний оператор схрещування (поглинаюче схрещування, greedy crossover) був запропонований в 1985 році Д. Грефенстеттом у співавторстві з іншими вченими для розв’язку задачі комівояжера. Це евристичний оператор схрещування, орієнтований на використання знань про об'єкт. Ідея побудови “жадібного” алгоритму полягає в наступному. На кожному кроці послідовно вибираються кращі елементи із множини наявних, тобто рішення, що поліпшують цільову функцію, причому таким чином, щоб не порушувати діючих обмежень. Генерація нащадків відбувається за рахунок вибору кращих ділянок батьківських хромосом і їхнього наступного сполучення.

Схема роботи жадібного оператора схрещування може змінюватися залежно від характеру розв'язуваних задач.

Послідовність виконання жадібного оператора схрещування наведена нижче.

Крок 1. Обчислити значення цільової функції у відібраних для

схрещування хромосом: *f*(*H*1) і *f*(*H*2), *H*1 = {*h*11, *h*12, …, *h*1n},

*H*2 = {*h*21, *h*22, …, *h*2n}, 

Крок 2. Встановити: *j* = 1. Випадковим чином вибрати початкову точку

для генерації хромосоми-нащадка: *pj* = rand(1,*n*).

Крок 3. Встановити temp = *pj*; *j* = *j* + 1.

Крок 4. Визначити наступну точку хромосоми-нащадка:  , де  , ; – номер гена хромосоми *H*v , що відповідає значенню temp.

Крок 5. У випадку, якщо хромосома-нащадок складена повністю

(*j* = *n*), перейти на крок 8.

Крок 6. Виконати перевірку на передчасне замикання циклу: *pj* = *pd*,

. У випадку передчасного замикання циклу збільшити шлях за

рахунок включення гена, обраного випадковим образом із числа ще не

включених.

Крок 7. Виконати перехід на крок 3.

Крок 8. Кінець.

Практика показує, що застосування жадібного оператора схрещування підвищує швидкість збіжності розв’язку, але в той же час це сприяє зменшенню розмаїтості популяції, що веде до її швидкого виродження, а також зниженню можливостей виходу з локальних оптимумів.

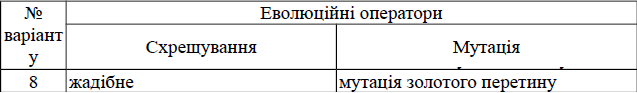
* *Мутація золотого перетину*

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**

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

Таблиця 1. Параметри еволюційного пошуку для виконання завдання

****

**Код програми**

clc;clear;close all;

nn=15; % кількість міст

asz=10; % розмір області asx x asz

ps=200; % чисельність популяції

ng=100; % Кількість поколінь

pm=0.05; % імовірність мутації обміну 2 випадкових міст на шляху (per gene, per genration)

pm2=0.02; % імовірність мутації обміну 2 частин шляху (per gene, per genration)

pmf=0.08; % імовірність мутації обміну випадкової частини шляху

r=asz\*rand(2,nn); % випадкове розподілення міст

dsm=zeros(nn,nn); % матриця відстаней

for n1=1:nn-1

r1=r(:,n1);

for n2=n1+1:nn

r2=r(:,n2);

dr=r1-r2;

dr2=dr'\*dr;

drl=sqrt(dr2);

dsm(n1,n2)=drl;

dsm(n2,n1)=drl;

end

end

% Почати з випадкових замкнених шляхів:

G=zeros(ps,nn);

for psc=1:ps

G(psc,:)=randperm(nn);

end

figure('units','normalized','position',[0.05 0.2 0.9 0.6]);

subplot(1,2,1);

% вивід найкращого шляху:

hpb=plot(NaN,NaN,'r-');

ht=title(' ');

hold on;

% вивід ділянок вузла

for n=1:nn

text(r(1,n),r(2,n),num2str(n),'color',[0.7 0.7 0.7]);

end

plot(r(1,:),r(2,:),'k.'); % виводити міста як чорні точки

axis equal;

xlim([-0.1\*asz 1.1\*asz]);

ylim([-0.1\*asz 1.1\*asz]);

subplot(1,2,2);

pthd=zeros(ps,1); %довжина шляху

for gc=1:ng % цикл поколінь

% знайти довжину шляху:

for psc=1:ps

Gt=G(psc,:);

pt=0; % підсумовування довжини шляху

for nc=1:nn-1

pt=pt+dsm(Gt(nc),Gt(nc+1));

end

% останній і перший:

pt=pt+dsm(Gt(nn),Gt(1));

pthd(psc)=pt;

end

min(pthd)

[mbp(gc) bp]=min(pthd);

Gb=0;

G(bp,:);

Gb=G(bp,:); % найкращий шлях

% оновити найкращий шлях на фігурі:

if mod(gc,5)==0

set(hpb,'Xdata',[r(1,Gb) r(1,Gb(1))],'YData',[r(2,Gb) r(2,Gb(1))]);

set(ht,'string',['generation: ' num2str(gc) ' best path length: ' num2str(pthd(bp))]);

plot(mbp)

title('best path length');

xlabel('generation number');

ylabel('best path value');

drawnow;

end

% схрещування:

Gch=zeros(ps,nn); % діти

for psc=1:(ps/2)

i1=1+2\*(psc-1);

i2=2+2\*(psc-1);

g1=G(i1,:); %перший ген

g2=G(i2,:); %інший ген

for i3=0:1

p1=0;

p1(1)=randi([1,nn]); %1 krok

j=1;

while j<nn

temp=p1(j);

if temp==nn

temp=0;

end

j=j+1;

fh1=dsm(p1(j-1),g1(temp+1)); %відстань між попереднім обраним геном і геном 1го батька

fh2=dsm(p1(j-1),g2(temp+1)); %відстань між попереднім обраним геном і геном 2го батька

j21=0; %чи елемент g1(temp+1) вже в нащадку

j22=0; %чи елемент g2(temp+1) вже в нащадку

j23=0; %чи вибрано ген для нащадка

for j1=1:j-1 %чи 2 гени батьків є в нащадку

if p1(j1)==g1(temp+1)

j21=1;

end

if p1(j1)==g2(temp+1)

j22=1;

end

end

if [j21==0,j22==1] %якщо 1 нема, 2 є

p1(j)=g1(temp+1);

j23=1;

end

if [j22==0,j21==1] %якщо 2 нема, 1 є

p1(j)=g1(temp+1);

j23=1;

end

if [fh1<fh2,j21+j22==0] %якщо 1 ген менший то його обираєм

p1(j)=g1(temp+1);

elseif (j21+j22)==0 %якщо 2 ген менший то його обираєм

p1(j)=g2(temp+1);

else %якщо дані гени батьків вже є в нащадку до виираєм випадковим чином із ще не обраних

j1=1;

while j1>0

j1=0;

j2=randi([1,nn]);

for j3=1:j-1

if p1(j3)==j2

j1=1;

end

end

end

p1(j)=j2;

end

end % while

Gch(i1+i3,:)=p1;

end

end %for psc=1:(ps/2)

G=0;

G=Gch;

% мутація золотого перетину:

for psc=1:ps

if rand<pm

n=ceil(nn\*0.61803);

G(psc,n:n+1)=fliplr(G(psc,n:n+1));

end

end

G(1,:)=Gb; % елітарність

end

**Результат виконання**

Імовірність мутації = 0.05.

**№1.** Кількість міст = 15. Чисельність популяції = 50. Кількість поколінь = 5.

Найкраща відстань між містами = 187.1099

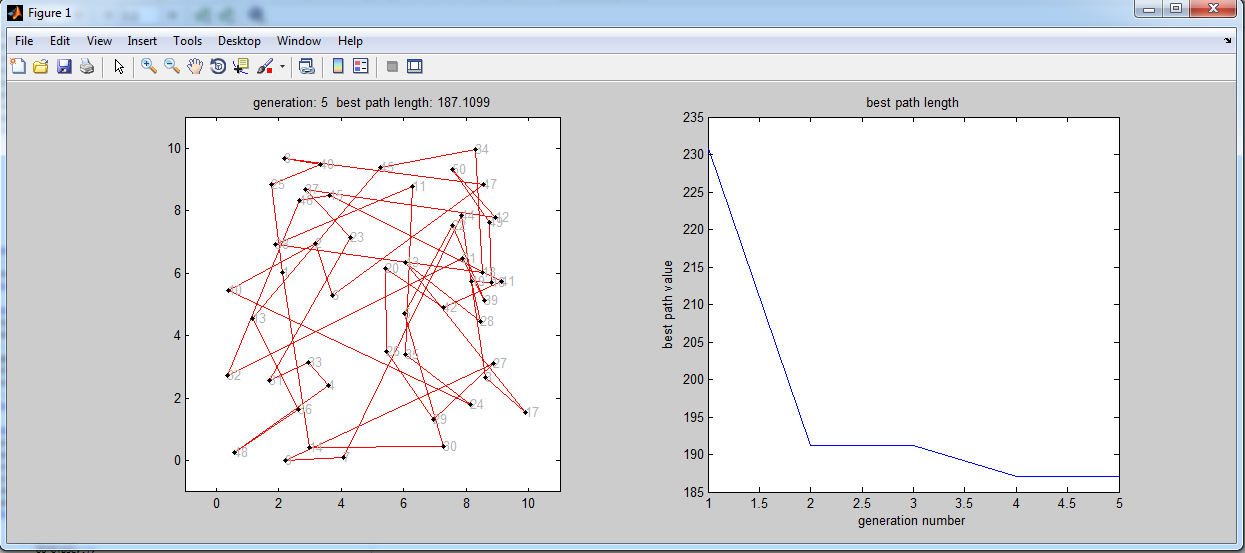


Рис.1. Результат виконання програми №1.

**№2.** Кількість міст = 30. Чисельність популяції = 100. Кількість поколінь = 20.

Найкраща відстань між містами = 184.1356

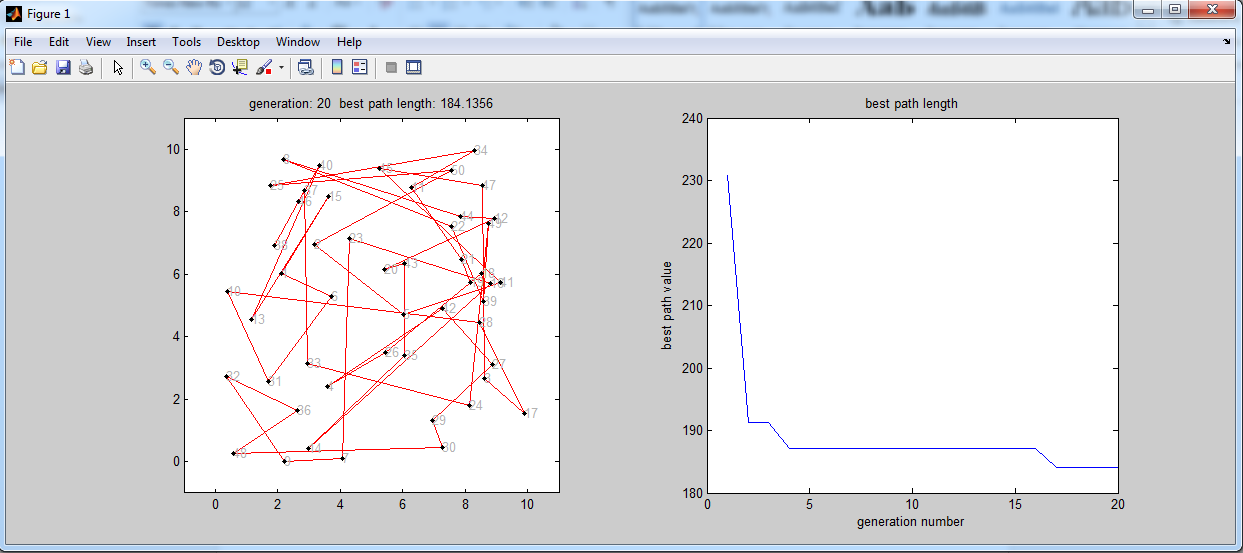


Рис.2. Результат виконання програми №2.

**№3.** Кількість міст = 50. Чисельність популяції = 200. Кількість поколінь = 130.

Найкраща відстань між містами = 179.2847

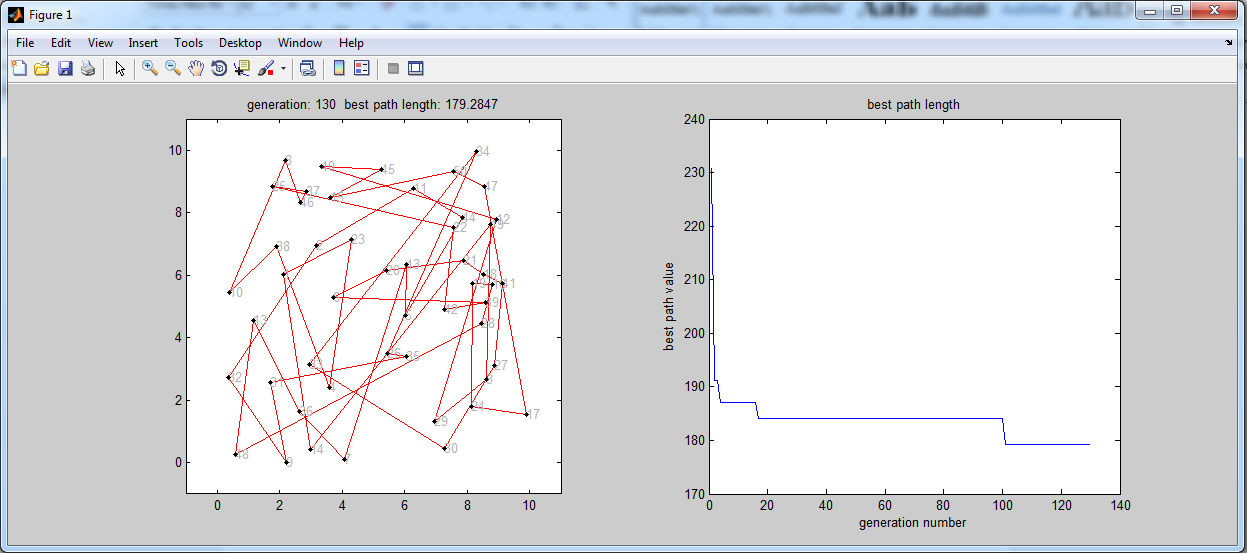


Рис.3. Результат виконання програми №3.

**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. В програмі реалізовано мутацію золотого перетину і жадібний оператор схрещування.