Прізвище: Жук

Ім’я: Богдан

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.

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(t·L), де t = 0.61803. В результаті застосування оператора мутації золотого перетину хромосома H={h1,h2,...,hD,hD+1,...,hL} перетворюється у хромосому H={h1,h2,...,hD+1,hD,...,hL}.

**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

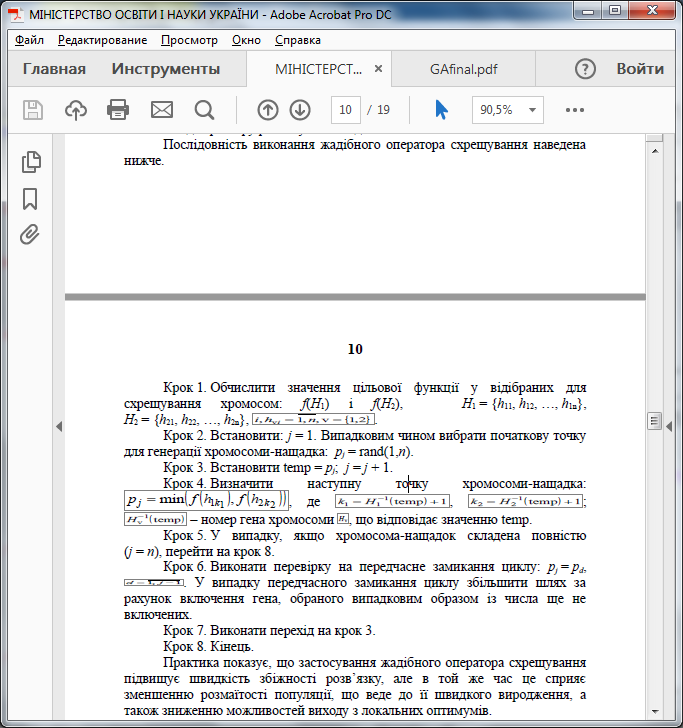
залежно від характеру розв'язуваних задач.

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},

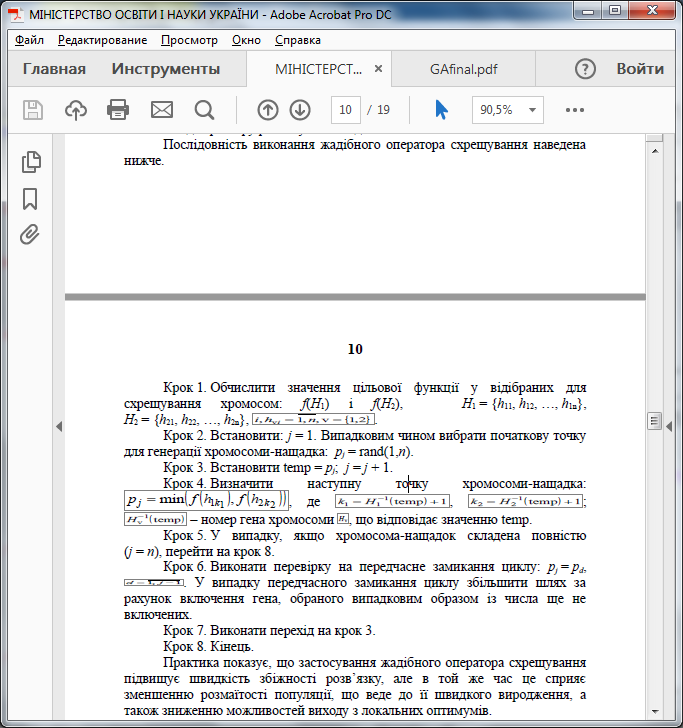
*H*2 = {*h*21, *h*22, …, *h*2n}, *i*,*h*v*i* = 

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:



*H* – 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.

**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 |
| 8 | жадібне | мутація золотого перетину |

**Код головного файлу програми:**

nn=40; % кількість міст

asz=10; % розмір області

ps=500; % чисельність популяції

ng=240; % Кількість поколінь

pm=0.1; % імовірність мутації двох міст

pm2=0.1; % імовірність мутації двох частин шляху

pmf=0.5; % імовірність мутації 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

plot(1);

% вивід найкращого шляху:

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

ht=title(' ');

hold on;

% вивід ділянок вузла

for n=1:nn

text(r(1,n),r(2,n),num2str(n));

end

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

axis equal;

%встановлення розміру графіка

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

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

pthd=zeros(ps,1); %довжина шляху

p=zeros(ps,1); % імовірності

%цикл поколінь

for gc=1:ng

% 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

ipthd=1./pthd; % зворотня довжина шляху

p=ipthd/sum(ipthd); % імовірності

[mbp bp]=max(p);

Gb=G(bp,:); % найкращий шлях

% Оновлення фігури

if mod(gc,1)==0

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

set(ht,'string',['генерацій: ' num2str(gc) ' найкраща довжина шляху: ' num2str(pthd(bp))]);

drawnow;

end

% схрещування:

ii=greedy(ps,p); % іі - гени номери генів міст, що будуть використані

Gc=G(ii,:); % гени для схрещування

Gch=zeros(ps,nn); % діти

for prc=1:(ps/2) % підрахунок пар

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

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

g1=Gc(i1,:); % перший ген

g2=Gc(i2,:); % другий ген

cp=ceil((nn-1)\*rand);% точка схрещування, випадкове число в інтервалі [1; nn-1]

% двоє дітей:

g1ch=mutation\_gold(g1,g2,cp);

g2ch=mutation\_gold(g2,g1,cp);

Gch(i1,:)=g1ch;

Gch(i2,:)=g2ch;

end

G=Gch;

for psc=1:ps

if rand<pm

rnp=ceil(nn\*rand);

rpnn=randperm(nn);

ctp=rpnn(1:rnp);

Gt=G(psc,ctp);

Gt=Gt(randperm(rnp));

G(psc,ctp)=Gt;

end

end

% мутація золотого перетину:

for psc=1:ps

if rand<pm2

cp=1+ceil((nn-3)\*rand); % діапазон

G(psc,:)=[G(psc,cp+1:nn) G(psc,1:cp)];

end

end

for psc=1:ps

if rand<pmf

n1=ceil(nn\*rand);

n2=ceil(nn\*rand);

G(psc,n1:n2)=fliplr(G(psc,n1:n2));

end

end

G(1,:)=Gb;

end

**Результат виконання лабораторного завдання.**

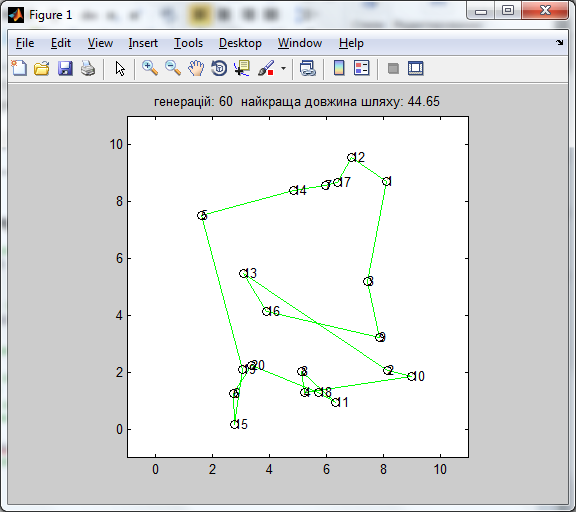


Рис 1. Розв’язок при параметрах: кількість генерацій 60, кількість міст 20

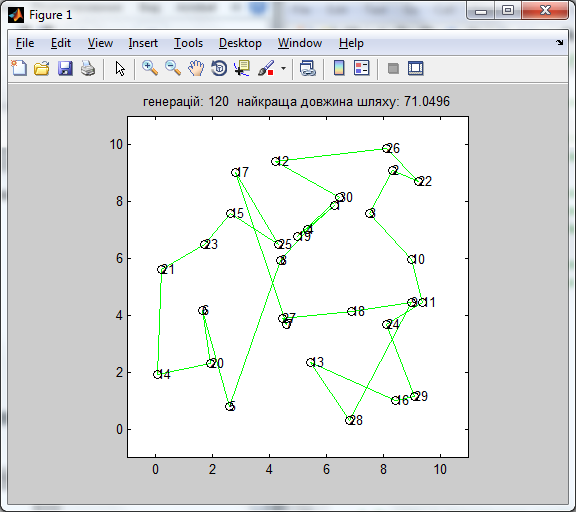


Рис 2. Розв’язок при параметрах: кількість генерацій 120, кількість міст 30

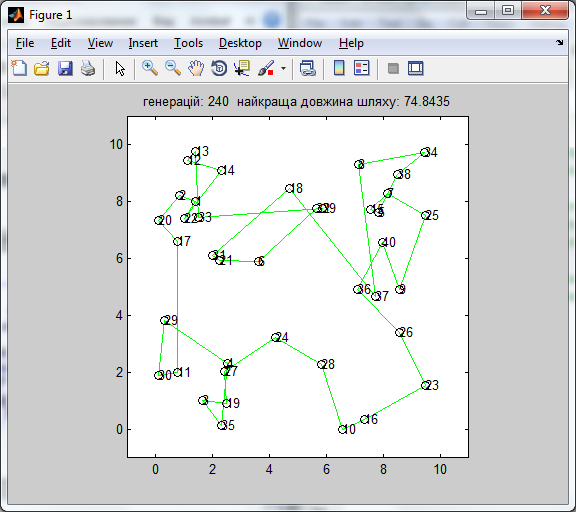


Рис 3. Розв’язок при параметрах: кількість генерацій 240, кількість міст 40

**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.