عملیات ژنتیکی:

کدگذاری واقعی GA شبیه سازی عملگر متقاطع (SBX) دودویی برای ادغام و جهش چند جمله ای استفاده می شود.

۱. شبیه سازی عملگر متقاطع. شبیه سازی متقاطع باینری در طبیعت مشاهده شده و در زیر آورده شده است.

$$c_{1,k} = \frac{1}{2}\left[(1-\beta_k)_{p_{1,k}} + (1+\beta_k)_{p_{2,k}}\right]$$

$$c_{2,k} = \frac{1}{2}\left[(1+\beta_k)_{p_{1,k}} + (1-\beta_k)_{p_{2,k}}\right]$$

در اینجا $c_{i,k}$ ، i^{th} کوچک با جزء k^{th} می باشد، $p_{i,k}$ منبع انتخاب شده که β_k (≥ 0) می باشد یک نمونه از عدد تصادفی تولید شده با داشتن چگالی زیر می باشد

$$p(\beta) = \frac{1}{2}(\eta_c + 1)\beta^{\eta_c}, \qquad if\ 0 \leq \beta \leq 1$$

$$p(\beta) = \frac{1}{2}(\eta_c + 1)\frac{1}{\beta^{\eta_c+2}}, \qquad if\ \beta > 1$$

این توزیع را می توان از یک عدد تصادفی نمونه برداری یکنواخت u بین $(0,1)$ به دست آورد. η_c شاخص توزیع برای تقاطع می باشد. بنابراین

$$\beta(u) = (2u)^{\frac{1}{(\eta+1)}}$$

$$\beta(u) = \frac{1}{[2(1-u)]^{\frac{1}{(\eta+1)}}}$$

۲. تحول چند جمله ای

$$c_k = p_k + \left(p_k^u - p_k^l\right)\delta_k$$

در اینجا c_k کودک و p_k خانواده می باشد با p_k^u حد بالا در اجزاء خانواده می باشد، p_k^l حد پایین و δ_k تغییرات کوچک که از توزیع چند جمله ای بصورت زیر محاسبه شده است

$$\delta_k = (2r_k)^{\frac{1}{\eta_m+1}} - 1, \qquad if\ r_k < 0.5$$

$$\delta_k = 1 - [2(1-r_k)]^{\frac{1}{\eta_m+1}}, \qquad if\ r_k \geq 0.5$$

r_k یک عدد تصادفی نمونه برداری یکنواخت بین $(0,1)$ و η_m شاخص تحول توزیع می باشد.

کدهای متلب برای الگوریتم ژنتیک:

```
function f = genetic_operator(parent_chromosome,pro,mu,mum);
[N,M] = size(parent_chromosome); switch pro
case 1
M = 2;
V = 6;
case 2
M = 3;
V = 12;
end p = 1; was_crossover = 0; was_mutation = 0; l_limit = 0;
u_limit = 1; for i = 1 : N
if rand(1) < 0.9
child_1 = [];
child_2 = [];
parent_1 = round(N*rand(1));
if parent_1 < 1
parent_1 = 1;
end
parent_2 = round(N*rand(1));
if parent_2 < 1
parent_2 = 1;
16 ARAVIND SESHADRI
end
while isequal(parent_chromosome(parent_1,:),parent_chromosome(parent_2,:))
parent_2 = round(N*rand(1));
if parent_2 < 1
parent_2 = 1;
end
end
parent_1 = parent_chromosome(parent_1,:);
parent_2 = parent_chromosome(parent_2,:);
for j = 1 : V
%% SBX (Simulated Binary Crossover)
% Generate a random number
u(j) = rand(1);
if u(j) <= 0.5
bq(j) = (2*u(j))^(1/(mu+1));
else
bq(j) = (1/(2*(1 - u(j))))^(1/(mu+1));
end
child_1(j) = ...
0.5*(((1 + bq(j))*parent_1(j)) + (1 - bq(j))*parent_2(j));
child_2(j) = ...
0.5*(((1 - bq(j))*parent_1(j)) + (1 + bq(j))*parent_2(j));
if child_1(j) > u_limit
```

```
child_1(j) = u_limit;
elseif child_1(j) < l_limit
child_1(j) = l_limit;
end
if child_2(j) > u_limit
child_2(j) = u_limit;
elseif child_2(j) < l_limit
child_2(j) = l_limit;
end
end
child_1(:,V + 1: M + V) = evaluate_objective(child_1,pro);
child_2(:,V + 1: M + V) = evaluate_objective(child_2,pro);
was_crossover = 1;
was_mutation = 0;
else
parent_3 = round(N*rand(1));
if parent_3 < 1
parent_3 = 1;
end
% Make sure that the mutation does not result in variables out of
% the search space. For both the MOP's the range for decision space
% is [0,1]. In case different variables have different decision
% space each variable can be assigned a range.
child_3 = parent_chromosome(parent_3,:);
for j = 1 : V
r(j) = rand(1);
if r(j) < 0.5
delta(j) = (2*r(j))^(1/(mum+1)) - 1;
else
delta(j) = 1 - (2*(1 - r(j)))^(1/(mum+1));
end
child_3(j) = child_3(j) + delta(j);
if child_3(j) > u_limit
child_3(j) = u_limit;
elseif child_3(j) < l_limit
child_3(j) = l_limit;
end
end
child_3(:,V + 1: M + V) = evaluate_objective(child_3,pro);
was_mutation = 1;
was_crossover = 0;
end
if was_crossover
child(p,:) = child_1;
child(p+1,:) = child_2;
was_cossover = 0;
p = p + 2;
```

```
elseif was_mutation
child(p,:) = child_3(1,1 : M + V);
NSGA-II 17
was_mutation = 0;
p = p + 1;
end
end f = child;
}
```