
بسم الله الرحمن الرحيم

موضوع پروژه

بررسی الگوریتم SPEA-II

مقطع و رشته تحصیلی

کارشناسی ارشد هوش مصنوعی و رباتیکز

استاد مربوطه

دکتر سید جلال الدین موسوی راد

گرد آورنده

مهدیه سادات فخاری

بهار ۹۶

❖ چکیده:

الگوریتم SPEA در سال ۲۰۰۱ ارائه شده است. دو سال بعد الگوریتم SPEA-II ارائه شد. در ابتدا پرداخته می شود به توضیحی از SPEA ساده (strength pareto evolutionary algoritm) یک الگوریتم تکاملی می باشد از آپراتور های ژنتیکی استفاده می کنند در واقع یک جور الگوریتم ژنتیک چند هدفه می باشد. مهم ترین بخشی که وجود دارد در این الگوریتم شدت pareto است که در SPEA و SPEA-II این تعریف متفاوت می باشد. در واقع این تعریف به ما نشان می دهد یک پاسخ چه مقدار به pareto نزدیک است اما نسبی است مطلق نمی باشد. در الگوریتم SPEA-II یک سری مشکاتی از جمله: مشکل خوشه بندی، غلبه شدن یک سری پاسخ ها توسط یک گروه ثابت، فاقد عامل ثانویه بودن. در الگوریتم SPEA-II مشکلات را حل می کند. به طور کلی مراحل SPEA-II بیان شده است به شرح ذیل می باشد:

قبل از بررسی مراحل تعارف اولیه مورد نیاز است. که به این صورت بیان می شود.

جمعیت اصلی در t = Pt	آرشیو در تکرار t ام: $\bar{P}t$	N= |Pt| تعداد اعضای Pt	$\bar{N}$= $|\bar{P}t|$تعداد ائلمان های $\bar{P}t$

❖ مراحل اصلی:

۱- آماده سازی : تولید P0 و$\bar{P}=\emptyset$

۲- محاسبه برازندگی pt و $\bar{P}t$

۳- انتخاب : اعضای نامغلوب $Pt \cup \bar{P}t$ را به $\bar{P}t+1$ منتقل می کنیم . ($\bar{P}t+1$ پیش نویس آرشیو مرحله بعد می باشد امکان اینکه کمتر، بیشتر، برابر بودن وجود دارد.).

اگر $|Pt+1|<N$ باشد، آنگاه آن را پر می کنیم ب استفاد از اعضای مغلوب Pt.

اگر $|Pt+1|>N$ باشد، اعضای اضافی با روش خاصی حذف می شوند.

۴- در صورتی که شرایط توقف محقق شده باشند، اعضای نامغلوب $Pt+1$ به عنوان پاسخ های نهایی معرفی می شوند.

۵- با استفاده از تورنومتر باینری والدین را از Pt+1 انتخاب می کنیم. (تورنومتر باینری به این معنا می باشد که: برا ساس فیتنس است به تصادف دو عضو از جمعیت اصلی انتخاب می کنیم و اونی را که f کمتری دارد انتخاب می شود مجدد یک تورنومتر دیگر برداشته انقدر این کار را انجام می دهیم تا یک mating pool ایجاد کنیم بعد روی تمام آپراتور ها تقاطع و جهش اعمال می شود).

۶- عملیات جهش و تقاطع بر روی والدین انجام می شوند و Pt+1 ایجاد می شود.

۷- یک واحد به شمارنده اضافه می کنیم.

۸- تکرار از مرحله ۲.

❖ محاسبه برازندگی در SPEA-II :

$$S(i)=\frac{|\{j \mid j \in Pt \cup \bar{P}t \ \& \ I \text{ dom } j\}|}{i \in \bar{P}t \cup Pt}$$

strength برای همه عضو ها به این گونه تعریف می شود یک مجموعه ای را در نظر می گیریم مجموعه همه j هایی که، این j ها یا عضو جمعیت هستند یا عضو ارشیو و i ، dominate می کند j یعنی اعضایی از جمعیت یا آرشیو توسط j غلبه می شوند.

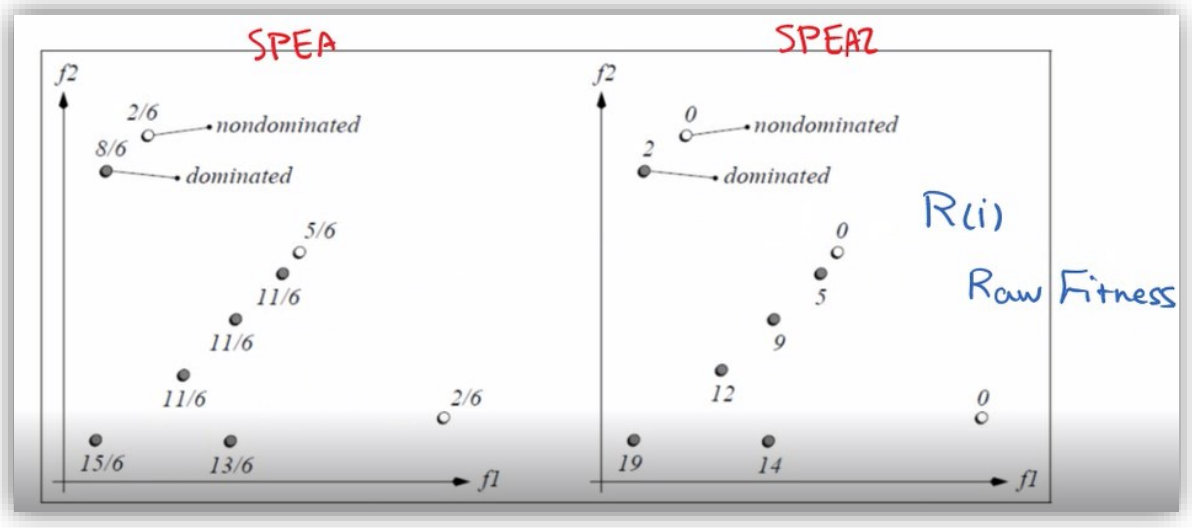
مفهوم s(i) یعنی strength ، i تعداد ائلمان هایی هست که چه در جمعیت اصلی چه در جمعیت آرشیو i بر آنها غلبه دارد اما شامل مساوی ها نمی شود.

❖ توضیحی از برازندگی خام(Raw fitness) :

دلیل اینکه برازندگی خام را بهش نسبت می دهیم این است که اصل کار شکل گرفته است اما قسمتی از آن تکمیل نیست و قرار است ما ان قسمت را تشکیل بدهیم به همین منظور R(i) را تعریف می کنیم.

$$R(i) = \sum_{\substack{k \in P_t \cup \bar{P}_t \\ k \text{ domi}}} S(k)$$

یعنی افرادی را که i را dominate می کنند s آنها را با هم جمع کن.و این را می دانیم که Raw fitness حتما یک عدد صحیح است.



شکل ۱- نمایی از Raw fitnes

برای پیاده سازی الگوریتم یک سری مسائلی برای شناخت این الگوریتم وجود دارد که یکی از آنها MOP2 می باشد. دو تابع هدف یا همان CostFunction وجود دارد .

که دو تابع هدف به صورت زیر بیان شده است:

$F1(x)=1-\exp[-\sum_{i=1}^{n}(xi-\frac{1}{\sqrt{n}})^2]$

$F2(x)=1-\exp[-\sum_{i=1}^{n}(xi+\frac{1}{\sqrt{n}})^2]$

فنوتایپ

$X=(x1,x2,\ldots\ldots,xn)\in |Rn$ ⟵ ژنوتایپ $-4 < xi < +4$

بر خلاف نمایش استانداری که در الگوریتم های تک هدفه داریم پاسخ ها را در فضای ژنوتایپ نشان می دهیم در اینجا پاسخ ها در فضای فنوتایپ نشان داده می شوند.

❖ کد های پیاده سازی شده MOP2 در محیط متلب:

در ابتدا تابع که خروجی آن را با z که تابع هدف باشد نشان می دهد و MOP2 اسم تابع ، (X) هم ورودی می باشد. بنا به گفته قبلی دو مولفه قرار شد داشته باشیم اولین مولفه f1 و دومین مولفه f2 و به صورت عمودی بردار را بر می گردانیم.

مقدار n برابر با numel (x) که x همان ورودی های ما است. Z برابر شده است با همان مقدار تابع f1,f2 با تفاوت اینکه

```
function z=MOP2(x)
n=numel(x);
z=[1-exp(-sum((x-1/sqrt(n)).^2))
    1-exp(-sum((x+1/sqrt(n)).^2))];
    end
```

این را ذخیره کرده و تابع MOP2 ما ایجاد شده و برای فراخوانی آماده می باشد.که می شود برای مسائل بهینه سازی استفاده کرد و نشان داد که مسئله،یک مسئله بهیه سازی است.

❖ مراحل پیاده سازی SPEA-II در محیط متلب :

در ابتدا صفحه جدید باز کرده و از clc و clear و close all کدهارا شروع می کنیم که به ترتیب صفحه رو پاک می کند، حافظه را پاک می کند،نمودار اضافی و ... را می بندد.

برنامه را از نظر منطقی به چند قسمت تقسیم می کنیم :

۱- Problem Definition
۲- Settings SPEA-II
۳- Initialization
۴- Main Loop
۵- Results

❖ در قسمت اول Problem Definition ، تابع هدف یا CostFunction تابعی که قرارا است کمینه شود که برابر است با یک تابعی از (x) یا خروجی همان MOP2 (x) می باشد. تعداد متغیر های تصیمیم را با nVar نشان می دهیم که مقدار آن برابر با ۳ شده است. معمولا ساختاری افقی در نظر گرفته می شود که اینجا در حال حاضر به دلایلی به شکل عمودی در نظر می گیریم. VarSize برابر شده است به صورت [nVar 1] که به شکل بردار عمودی نشان داده شده. در واقع VarSize اندازه ماتریسی است که متغییر های مجهول ما هستند.
حد بالا و حد پایین که همان VarMin و VarMax می باشد را با یک عددی مشخص می کنیم که عدد تعیین شده در قسمت بالا از -۴ تا ۴ بوده است.

```
CostFunction = @(x) MOP2 (x);
nVar =3;            % Number of Decision Variables
VarSize = [nVar 1];   % Decision Variables Matrix Size
VarMin  = -4 ;         % Decision Variables Lower Bound
VarMax = 4;          % Decision Variables Upper Bound
```

- در قسمت دوم SPEA-II Settings ، تنظیمات مربوط به الگوریتم می باشد.که شامل اندازه جمعیت اصلی ، حلقه تکرار برای رفتن به جلو، و... . بیشترین تعداد تکرار های چرخه تکامل را با MaxIt مشخص می کنیم که در ایجا بابر با ۱۰۰ بار تکرار است. تعداد اعضای جمعیت اصلی با npop نمایش داده می شود که بر فرض مثال برابر با ۲۰ است و تعداد اعضای آرشیو nArchive برابر با ۱۰ است .در ادامه k را داریم همان پارامتر kNN می باشد که جذر npop, nArchive می باشد. احتمال انجام crossover در واقع چند درصد از اعضای جمعیت جدید که همان جمعیت قبلی است در اثر crossover ایجاد می شوند در این بخش pCrossover برابر با ۰.۷ قرار داده شده است، ۰.۷ از آن تعداد ۲۰ تایی که می خواهند تولید بشوند از crossover می باشد. تعداد عضویت آنها را با nCrossover نشان می دهیم ، برابر است با pCrossover*nPop، اما محدودیت ذاتی که وجود دارد این است که حتما باید تعداد اینها زوج باشد، بنابراین تقسیم بر ۲ می کنیم و برای اینکه روند شود کل آن را ضرب در ۲ می کنیم . در اینجا دو دلیل برای ایجاد اعضای جمعیت جدید داریم یعنی یا باید از طریق crossover یا از طریق Mutation ایجاد شوند.به همین دلیل درصدی که با crossover ایجاد می شوند و درصدی که با Mutation باید هر دو برابر ۱ باشد.برای محاسبه جمعیت Mutation، که با nMutation نمایش می دهیم برابر است با nPop-nCrossover . نوع انتخاب در این روش با استفاده از تورنومتر باینری می باشد.

```
MaxIt=۱00;      % Maximum Number of Iterations
nPop=۲۰;        % Population Size
nArchive=10;    % Archive Size
K= round( sqrt(nPop+nArchive));    % KNN Parameter
pCrossover=0.7;
nCrossover=round(pCrossover*nPop/2)*2;
pMutation=1-pCrossover;
nMutation=nPop-nCrossover;
```

- در قسمت سوم Initialization، در ابتدا آماده سازی و محاسبه برازندگی وجو دارد.اول آماده سازی را انجاد داده، برای هر پاسخی لازم است یک سری پارامتر هایی در نظر گرفته شود. مانند تابع هدف، مقدار فواصل ، مقدار فیتنس و ... . انتظاری که از یک فرد یا یک جواب انتظار داریم میریزیم داخل یک empty_individual به معنای اینکه یک فضای خالی را در نظر می گیریم .

empty_individual قبل از همه مستقل ترین خاصیت این Position است که در فضای جستجو دارد.و بعد از این مهم ترین خاصیت Cost می باشد. این دو در هر الگوریتم ثابت می باشد. در این الگوریتم SPEA-II این empty_individual یک شدت (strength) را دارد که با s نشان داده می شود در ضمن یک فیتس خامی را هم دارد (Raw) که با R نشان می دهیم ، و یک مجموعه ای از sigma ها وجود دارد که فواصل موجود هستند.و یک F را دارد که final fitnes ما می باشد. این ها همه خواص یک individual می باشد.که باعث می شود خاصیت ها به صورت فرضی داخل یک empty_individual به صورت ساخت یافته داشته باشیم.

```
Empty_individual.Position=[];
empty_individual.Cost=[];
empty_individual.S=[];
empty_individual.R=[];
empty_individual.sigma=[];
empty_individual.sigmaK=[];
empty_individual.D=[];
empty_individual.F=[];
```

این هارو باید تکثیر کنیم و از آن ها جمعیت بسازیم که به صورت عمودی تکثیر را انجام می دهیم .

Pop ها ایجاده شده اند خالی هستند که باید آنها را پر کنیم. یک حلقه for ایجاد کرده که در این حلقه for برای مقدار های ما از حد بالا و حد پایین و اندازه ساختاری استفاده نموده است.

```
Pop=repmat(empty_individual,nPop,1);
for i=1:nPop
pop(i).Position=unifrnd(VarMin,VarMax,VarSize);
pop(i).Cost=CostFunction(pop(i).Position);
end
archive=[];
```

- در قسمت چهارم Main Loop ، این قسمت از یک دستور for استفاده کرده.. در مرحله بعد فیتنس هارو محاسبه نمونده . فیتنس اول s می باشد که s تعداد افرادی که در این بایگانی هستند که j هایی توسط i غلبه می شوند در واقع شدت و قدرت را نشان می دهد.چون همیشه داریم آرشیو را در جمعیت ادغام می کنیم این جمعیت بزرگ را با Q مشخص میکنیم. در مرحله بعد به ازای j هایی که در جمعیت هستند با i ها مقایسه کنیم، که مجدد از حلقه for استفاده می شود. برای اینکه هر بار داره j و i یکدیگر را بررسی میکنند و یک کار اضافی است باید مسئله را به این صورت تعریف کنیم که بیا هر بار با بعدی مقایسه رو انجام بده که با i+1 مشخص نموده ایم.حال باید بررسی شود اگر پاسخ i دارد پاسخ j را غلبه می کند آنگاه به strength، i یک واحد اضافه می کند، و اگر برعکس است یک واحد به strength، j اضافه می کند. در قسمت بعد تابع Dominates را به صورت کامل بیان نموده ایم اما در این بخش تابع را فراخوانی کرده به این صورت که اگر پاسخ Q(i) Dominates می کند پاسخ Q(j) را آنگاه عضو i از جمعیت ادغام شده s برابر است با یک واحد بیشتر در غیر این صورت اگر بر عکس شد یک واحد به j اضافه می کند. در ادامه ماتریسی به اسم dom تعریف می کنیم برای محاسبات سریه Raw fitness ، برای i و j ,dom را برابر یک قرار می دهیم.برای استفاده از این ماتریس از یک حلقه for استفاده می کنیم که بررسی می کند s مربوط به ائلمان هایی که i را دارند Dominates می کنند.

```
For it=1:MaxIt
```

```
Q=[pop
    archive];
    nQ=numel(Q);
    dom=false(nQ,nQ);
    for i=1:nQ
     Q(i).S=0;
      end
     for i=1:nQ
     for j=i+1:nQ
     if Dominates(Q(i),Q(j))
       Q(i).S=Q(i).S+1;
     dom(i,j)=true;
     elseif Dominates(Q(j),Q(i))
    Q(j).S=Q(j).S+1;
     dom(j,i)=true;
   end
   end
  end
   end
   for i=1:nQ
   Q(i).R=sum(S(dom(:,i)));
   end
```

مفهوم غلبه را بیان می کنیم و برای استفاده از آن تابعی برایش تعریف می کنیم تا در صورت نیاز در هر قسمت برنامه از آن استفاده کنیم .

❖ تعریف تابع Dominates به صورت زیر می باشد. می خواهیم ببینیم x، y را غلبه می کند یا خیر ؟

بافرض اینکه x و y مقادیر خود همان تابع هدفشان را داشته باشند. اگر x ما یک isstruct هست و فیلدی به اسم cost هم داخلش است آنگاه x همان cost می باشد. برای y هم به همین صورت می باشد. در اخر می گوییم x ، y غلبه می کند اگر و فقط اگر هیچ جا بدتر از آن نباشد و در یک جا حداقل بهتر باشد.

```
function b=Dominates(x,y)
   if isstruct(x) && isfield(x,'Cost')
      x=x.Cost;
   end
   if isstruct(y) && isfield(y,'Cost')
      y=y.Cost;
   end
   b=all(x<=y) && any(x<y);
end
```

- در این بخش می خواهیم محاسبه فواصل را بررسی کنیم که x برابر شد با جمع همه ی موقعیت ها.یک ماتریسی بزرگ به اسم SIGMA برابر است با یم تابع آماری به اسم pdist2 که شکل های گوگونای دارد اما در اینجا از seuclidean استفاده کرده ایم که استاندار اقلیدوسی است به این معنا گه متغیر ها در انحراف معیارشان تقسیم می شوند، باعث به وجو آمدن نرمال شدن آماری اتفاق افتد.به فرض مثال اگر تابع هدف ما پراکنده باشد با این ورش پاسخ بهتری را خواهیم به دست آورد. که بعد ها در مرحله

بعد هم از ان استفاده شده است. همسایگی در این بخش را می شود هم در فضای ژنوتیپ در نظر گرفت هم در فضای فنوتیپ اما چون ذات این الگوریتم این همسایگی در فضای فنوتیپ در نظر گرفته می شود.بنابراین cost را در نظر میگیریم و xهای ما تبدیل به z می شوند. بعد از محاسبه SIFMA ما یک ماتریس ۲۰*۲۰ می باشد.و برای مرتب سازی این SIGMA از sort استفاده می کنیم. محاساتی که در قسمت بالا تعرق شده اند را در اینجا در یک حلقه for آنهار انجام می دهیم.و بعد از اجرا خواهیم دید مقدار هایی که درقبل از نوشتن این for خالی بوده اند الان داای مقدار شده اند.

```
Z=[Q.Cost]';
SIGMA=pdist۲(Z,Z,'seuclidean');
SIGMA=sort(SIGMA);
for i=۱:nQ
    Q(i).sigma=SIGMA(:,i);
    Q(i).sigmaK=Q(i).sigma(K);
    Q(i).D=۱/(Q(i).sigmaK+۲);
    Q(i).F=Q(i).R+Q(i).D;
end
```

- در این قسمت nND همان غلبه نشده های ما هستند . بررسی را انجام میدهد که اگر کمتر یا بیشتر بود چه اتفاقی رخ بدهد در ابتدا nND<=nArchive بررسی کرده که اگر برقرار بود کافی است که این اعضای جمعیت Q را بر اساس فیتنس مرتب کنبم به این صورت که فیتنس هر چقدر کمتر باشد بهتر است در غیر اینصورت اگر بیشتر بود باید معیار ثانویه را در نظر بگیریم که این معیار ثانویه را از داخل SIGMA برداشته می شود. عامل مهم در مرتب سازی ما so می باشد زیرا Q را طبق so میچینیم در

واقع آرشیو جدید برابر می شود با nArchive عضو اول، رمانی این روند اتفاق می افتد که کمتر باشد اما اگر بیشتر باشد باید حذف کنیم.ایتدا بر اساس فواصلشون مرتب کرده بعد حذف را بر اساس SIGMA انجام می دهیم .در غیر اینصورت

```
nND=sum([Q.R]==0);
   if nND<=nArchive
      F=[Q.F];
      [F, SO]=sort(F);
      Q=Q(SO);
      archive=Q(1:min(nArchive,nQ));
else
```

- تعداد اعضای آرشیو را تا زمانی که بزرگتر است حلقه تکرار، تکرار می شود در ضمن این کار را بر اساس همسایه شماره ۲ یا k=2 زیرا خوش هم همسایه خودش حساب می شود بنابراین از همسایه شماره ۲ شروع می کنیم برای این کار همانطور که در قبل گفته شد ماتریس SIGMA داشتیم که هر ستون آن برای یک جواب بوده است و هر سطر آن برای یک رتبه است یعنی سطر اول نزدیک ترین همسایه می باشد .بنابراین برای ستون هایی که میخواهیم انتخاب شوند SIGMA را برابر صفر گذاشته. در اینجا چک می کنیم که بین اعضای یک سطر هر کدام که کمتر می باشند آنگاه باید حذف شود. همه رو چک میکنیم در هر مرحله تا جایی که می شود حدف را انجام میدهیم اما زمانی همه انحراف معیار های یک

سطر با هم برابر باشند نمی شود چیزی را حذف کرد بنابراین سراغ سطر بعدی رفته و چک می کنیم و تا جایی ادامه می دهیم که به سطر آخر نرسیده باشیم. برای پیدا کردن کمترین مقدار عاملی که مهم است برای ما محل وقوع آن می باشد و مقدار اون کمترین برای ما مهم نمی باشد که از طریق آرشیو عضو چندم است و ستون آن چندمی هست عملیات حذف را انجام می دهیم.اگر شماره تکرار به قسمت اخر رسیده خارج می شویم. زمانی که خارج شده ایم pareto front برابر اعضای آرشیو که فیتنس خام آنها برابر صفر می باشد.

```
SIGMA=SIGMA(:,[Q.R]= =0);
    archive=Q([Q.R]= =0);
     k=2;
     while numel(archive)>nArchive
     while min(SIGMA(k,:))==max(SIGMA(k,:)) && k<size(SIGMA,1)
      k=k+1;
       end
 [~, j]=min(SIGMA(k,:));
        archive(j)=[];
        SIGMA(:,j)=[];
     end
PF=archive([archive.R]==0); % Approximate Pareto Front
```

- در این قسمت می خواهیم تابع crossover را جداگانه پیاده سازی کرده تا در بخش اصلی بتوانیم آن را فراخوانی کنیم. که تابع تعریف شده به صورت زیر می باشد.

برای تعریف از یک تابع crossover محاسباتی استفاده می کنیم چون فضای محاسباتی پیوسته است. تابعی را تعریف می خواهیم تعریف کنیم که دارای دو ورودی والد و دو خروجی فرزند می باشد. در اثر crossover، x1,x2 تبدیل می شوند به y1, y2 که فرزندان ما هستند.این تابع محاسباتی چند پارامتر مختلف دارد بنابراین در یک پارامتری به اسم params عملیات جمع را ذخیره می کنیم. یک پارامتر مهم پارمتر ضریب خروج از بازه دو والد می باشد. با تغییر α بین ۰ و ۱ چون در یک بازه در حال تغییر هستند زیاذ جالب نمی باشد بنابراین اگر بخواهیم یکم بخواهیم بیشتر شود به جای این که در بازه ۰ و ۱ قرار دهیم α را به این صورت γ+1=>α=>γ قرار داده یعنی اجاده می دهد که به اندازه γ

کمتر شود و از ۱ بیشتر شود . با این کار باعث می شود که به والدینش محدود نشود.y خودش به عنوان یک پارامتر است. موضوع بعدی اینکه اینها به طور آزادانه می توانند هر نوع فرزندی را تولید کنند. بنابراینVarMin و VarMax داخل params آمده اند. در اینجا alpha که توضیح داده شده است را تعریف می کنیم یک سری عدد تصادفی در بازه گفته شده است با اندازه x1 .

در اینجا فضا چند بعدی می باشد باید برای هر بعد آنها مقداری باید نوشت. که y1 ما می شود بردار alpha ضرب در x1 می کند به همین ترتیب برای y2 هم همینطور می باشد. با این حال باید چک کنیم که y1 نباید کمتر از حد پایین باشد و نباید بیشتر از حد بالا باشد درواقع با این کار محدود می کنیم به همین صورت برای y2 هم داریم.

```
function [y1, y2]=Crossover(x1,x2,params)
  gamma=params.gamma;
    VarMin=params.VarMin;
    VarMax=params.VarMax;
    alpha=unifrnd(-gamma,1+gamma,size(x1));
    y1=alpha.*x1+(1-alpha).*x2;
    y2=alpha.*x2+(1-alpha).*x1;
    y1=min(max(y1,VarMin),VarMax);
    y2=min(max(y2,VarMin),VarMax);
```

- تابع دیگری را تعریف کرده به اسم Mutate که خروجی آن یک y و ورودی آن یک x می باشد. این mutationرا بر روی یک پاسخ انجام می دهد. در اینجا از یک mutation گوسی ساده برای نرمال سازی استفاده می کنیم که میانگین در این توزیع نرمال x می باش اما واریانس را بر اساس همان حد بالا و حد پایین تعریف می کنیم.انحراف معیار را برابر قرار می دهیم با درصدی از اختلاف حد بالا و حد پایین می باشد.

اگر بخواهیم به صورت گوسی باشد یک رندوم نرمال را با واریانس انحراف معیار sigms جمع می کنیم با x. به هر حال ما محدود سازی را هم در این قسمت انجام می دهیم . در حال حاضر دو عنصر اصلی که mutation , crossover را پیاده سازی نموده ایم.

```
function y=Mutate(x,params)
    h=params.h;
    VarMin=params.VarMin;
    VarMax=params.VarMax;
    sigma=h*(VarMax-VarMin);
    y=x+sigma*randn(size(x));
    y=min(max(y,VarMin),VarMax);
end
```

- توضیح تابع انتخاب تورنومتر باینری به این صورت است که خروجی آن عضوی از جمعیت می باشد .از اعضای یک جمعیت بر اساس یک معیاری مانند فیتنس میاییم انتخاب را انجام می دهیم بعد به ترتیب دو عضو از این جمعیت برداشته که نباید با هم برابر باشند یعنی نباید دو بار یکی را انتخاب کنیم برای این کار از تابع randsample استفاده می کنیم .

n تعداد ائلمان های جمعیت است و انتخاب شده هایمان را با I نمایش می دهیم که برابر است با randsample که دوتا را بدون تکرار انتخاب می کند. بعد بررسی را انجام می دهد که کدام یک بهتر است i1 یا i2 هرکدام بهتر است را انتخاب می کند.

```
function p=BinaryTournamentSelection(pop,f)
  n=numel(pop);
```

```
I=randsample(n,۲);
i۱=I(۱);
i۲=I(۲);
if f(i۱)<f(i۲)
 p=pop(i۱);
else
p=pop(i۲);
 end
end
```

- در اصل باید یک جمعیتی را داشته باشیم که والدین را از ان انخاب کند و با crossover جمعیت را تکمیل کند و یکی دیگر را داشته باشیم که mutation انجام بدهد به همین دلیل دو ستونه در نظر می گیریم به این صورت که به جای اینکه کلش را داشته باشیم تقسیم بر ۲ می کنیم. در ابتدا فاز crossover را داریم :

به این شکل است که در ایتدا یک جمعیت خالی را تولید می کنیم همانطور که گفته شده است ۰.۷ از ۲۰ تا جمعیت قرار است با crossover تولید شود. یعنی ۱۴ تا فرزند با ۷ تا Crossover تولید می شود. بنابر این یک for را گذاشته و از ۱ تا تعداد crossover قرار داده. برای هر crossover دو تا والد مورد نیاز است که این انتخاب والد را با باینری تورنومتر که تابع آن در قسمت بالا توضیح داده است را انجام می دهیم بر روی جمعیت آرشیو. والد اول را بررسی می کنیم که با استفاده از تورنومتر باینری از آرشیو نسبت به فیتنس آن یاF، F آن است که هر چقدر کمتر باشد مهم است این کار را انجام می دهد. والد دوم را به همین صورت انجام می دهیم. حال باید عملیات crossover را انجام بدهیم وقتی که انجام داده ایم باید قسمتcrossover_params این را مشخص کنیم که در قسمت setting SPEA-II ، یگ گاما داریم برابر با ۰.۱قرار می دهد و حد بالا و حد پایین را هم داریم.در ادامه داخل همین بخش

mutationرا هم می نویسیم به این صورت که mutation.params ان برابر است با ۰.۲ و یک حد بالا و یک حد پایین را هم داریم.

Crossover ٪

```
popc=repmat(empty_individual,nCrossover/۲,۲);
for c=۱:nCrossover/۲
    p۱=BinaryTournamentSelection(archive,[archive.F]);
    p۲=BinaryTournamentSelection(archive,[archive.F]) ;
    [popc(c,۱).Position,
popc(c,۲).Position]=Crossover(p۱.Position,p۲.Position,crossover_params);
    popc(c,۱).Cost=CostFunction(popc(c,۱).Position);
    popc(c,۲).Cost=CostFunction(popc(c,۲).Position);
end
```

- در این قسمت mutation را می خواهیم انجام دهیم. در ابتدا یک جمعیت خالی را تولید کرده و به صورت ستون تعریفش می کنیم به این شکل که popc را برابر popc(:) قرار می دهیم. یک حلقه for گذاشته و از تا mutation حلقه را می گذاریم. والد را با روش انتخاب تورنومتر باینری انجام داده از آرشیو بر اساس تابع F که هر چه کمتر باشد برای ما مهم است و بعد مراحل mutation انجام می شود.

```
popc=popc(:);
  popm=repmat(empty_individual,nMutation,1);
  for m=1:nMutation
    p=BinaryTournamentSelection(archive,[archive.F]);
    popm(m).Position=Mutate(p.Position,mutation_params);
    popm(m).Cost=CostFunction(popm(m).Position);
  end
```

و بعد از انجام این مراحل جمعیت جدیدمان را که ناشی از جمعیت crossover و mutation است را تولید می کند.

```
    % Create New Population
    pop=[popc
        popm];
end
```

- pareto front ما ده عضو بیشتر ندارد برای ترسیم آن در قسمت pareto front که pareto شکل می گیرد قسمتی به اسم figure باز کرده. بعد اگر هست که ایجاد کن اگر هیست تصویر را با همان انجام بده. جمع pareto front و cost را به شکل یک ماتریس ایجاد کرده که برابر قرارداده با PFC . و ترسیم آن به صورت ضرب در PFC1 را با PFC2 انجام می دهد.

```
PF=archive([archive.R]==0); % Approximate Pareto Front

    % Plot Pareto Front
    figure(1);
    PFC=[PF.Cost];
    plot(PFC(1,:),PFC(2,:),'x');
```

```
xlabel('f_1');
ylabel('f_2');
```

می توان در هر لحظه هم اطلاعات مربوط به ان را نمایش داد با قطعه کدی در قسمت ترسیم می توان این کار را انجام داد با دستور Display

```
%Display Iteration Information
disp(['Iteration ' num۲str(it) ': Number of PF members = ' num۲str(numel(PF))]);
```

- و در اخر قسمت نتیجه گیری ان کار هایی که در اصل برنامه انجام نداده ایم اگر بخواهیم انجام بدهیم در این قسمت تغییراتی را می توان اعمال کنیم. بر فرض مثال PFC را که در بالا داشتیم بخواهیم کمترین رو یا بیشترین، انحراف معیار، میانگین رو نمایش بدهیم که با دستور FOR این کار را انجام داده است و با دستور disp نمایش را صورت داده.

```
disp(' ');
for j=1:size(PFC,1)
    disp(['Objective #' num2str(j) ':']);
    disp(['      Min = ' num2str(min(PFC(j,:)))]);
    disp(['      Max = ' num2str(max(PFC(j,:)))]);
    disp(['    Range = ' num2str(max(PFC(j,:))-min(PFC(j,:)))]);
    disp(['    St.D. = ' num2str(std(PFC(j,:)))]);
    disp(['     Mean = ' num2str(mean(PFC(j,:)))]);
```

```
    disp(' ');
end
```