رایانش تکاملی

پروژه پایانی

علی قنبری ۹۶۰۲۵۱۸۱۸

نیمسال اول ۹۸-۹۷

۱. تعریف مساله

هدف اصلی بررسی کارایی الگوریتم های ژنتیک با کدینگ حقیقی و تکامل تفاضلی در حل مسائل بهینه سازی محدود شده با استفاده از توابع جریمه برای هریک از توابع زیر می باشد که مینیمم هریک را به دست آورده و چون مسئله کمینه سازی است کمترین مقدار بهترین جواب است.

در زیر توضیح مختصری از الگوریتم های گفته شده ارائه شده است:

الگوریتم ژنتیک:

الگوریتم های ژنتیک (به انگلیسی: Genetic Algorithm)، (با نماد اختصاری GA) تکنیک جستجویی در علم رایانه برای یافتن راه‌حل تقریبی برای بهینه‌سازی و مسائل جستجو است. الگوریتم ژنتیک نوع خاصی از الگوریتم‌های تکامل است که از تکنیک‌های زیست‌شناسی فرگشتی مانند وراثت و جهش استفاده می‌کند. این الگوریتم برای اولین بار توسط جان هلند معرفی شد.

در واقع الگوریتم‌های ژنتیک از اصول انتخاب طبیعی داروین برای یافتن فرمول بهینه جهت پیش‌بینی یا تطبیق الگو استفاده می‌کنند. الگوریتم‌های ژنتیک اغلب گزینه خوبی برای تکنیک‌های پیش‌بینی بر مبنای

رگرسیون هستند. در هوش مصنوعی الگوریتم ژنتیک (یا GA) یک تکنیک برنامه‌نویسی است که از تکامل ژنتیکی به عنوان یک الگوی حل مسئله استفاده می‌کند. مسئله‌ای که باید حل شود دارای ورودی‌هایی می‌باشد که طی یک فرایند الگوبرداری شده از تکامل ژنتیکی به راه‌حلها تبدیل می‌شود سپس راه حلها بعنوان کاندیداها توسط تابع ارزیاب (Fitness Function) مورد ارزیابی قرار می‌گیرند و چنانچه شرط خروج مسئله فراهم شده باشد الگوریتم خاتمه می‌یابد. الگوریتم ژنتیک بطور کلی یک الگوریتم مبتنی بر تکرار است که اغلب بخش‌های آن به صورت فرایندهای تصادفی انتخاب می‌شوند.

این الگوریتم‌ها از بخش‌های زیر تشکیل می‌شوند:

تابع برازش- نمایش- انتخاب- تغییر

الگوریتم تکامل تفاضلی:

الگوریتم تکامل تفاضلی (Differential Evolutuion) و یا به اختصار DE، یک الگوریتم بهینه سازی هوشمند و مبتنی بر جمعیت است که در سال ۱۹۹۵ توسط Storn و Price معرفی گردید. نسخه ابتدایی این الگوریتم برای حل مسائل پیوسته ارائه شده بود، اما به مرور زمان نسخه هایی از این الگوریتم ارائه شدند که برای حل مسائل بهینه سازی گسسته طراحی شده اند.

۲. حل مساله

در هر دو الگوریتم تولید جمعیت اولیه بصورت تصادفی و تعداد جمعیت ۵۰ و ماکزیمم تعداد نسل ۱۰۰۰ می باشد. در الگوریتم ژنتیک با کدینگ حقیقی عملگر آمیزش استفاده شده عملگر آمیزش محدب می باشد و برای انتخاب والدین برای تولید فرزند با عملگر آمیزش از چرخ رولت استفاده شده. برای عملگر جهش نیز از جهش پویا استفاده شده است. در الگوریتم تکامل تفاضلی عملگر آمیزش استفاده شده عملگر آمیزش دو جمله ای می باشد. برای تابع جریمه نیز از روش زیر استفاده می کنیم:

از فرم مجموع برای تابع ارزیابی به صورت زیر استفاده شده است:

$$Eval(x)=f(x)+p(t,x)$$

که در آن برای یافتن p(t,x) از رابطه زیر استفاده می شود:

$$P(t,x)=\rho^{\alpha}\ \sum_{i=1}^{m}\left(d_i(x)^{\beta}\right)$$

$$d_i(x)=\begin{Bmatrix}0\\|g_i(x)|\\|h_i(x)|\end{Bmatrix}$$

$$\rho = C * t$$

۲. توابع آزمون –مجموعه داده

به منظور بررسی کارایی الگوریتم‌های ژنتیک با کدینگ حقیقی و تکامل تفاضلی در حل مسائل بهینه سازی محدود شده، از سه تابع محک[1] با فرم زیر استفاده شده است.

- $\text{Min } f_1(x)=(x_1-2)^2+(x_2-1)^2$

  $\text{s.t. } g_1(x)=x_1-2x_2+1=0$

  $g_2(x)=x_1^2/4-x_2^2+1\geq 0$

  $0\leq x_i\leq 2 \quad (i=1,2)$

- $\text{Min } f_2(x)= 5.3578547x_3^2+0.8356891x_1x_5+37.293239x_1-40792.141$

  $\text{s.t. } 0\leq 85.334407+0.0056858x_2x_5+0.00026x_1x_4-0.0022053x_3x_5\leq 92$

  $90\leq 80.51249+0.0071317x_2x_5+0.0029955x_1x_2+0.0021813x_3^2\leq 110$

[1] Benchmark

$$20 \leq 9.300961+0.0047026x_3x_5+0.0012547x_1x_3+0.0019085x_3x_4 \leq 25$$

$$78 \leq x_1 \leq 102$$
$$33 \leq x_2 \leq 45$$
$$27 \leq x_3 \leq 45$$
$$27 \leq x_4 \leq 45$$
$$27 \leq x_5 \leq 45$$

- Min $f_3(x) = \sum_{j=1}^{10} xj \ (c_j+\ln(x_j/x_1+\ldots+x_{10})$

  s.t. $x_2+2x_2+2x_3+x_6+x_{10}=2$
  $x_4+2x_5+x_6+x_7=1$
  $x_3+x_7+x_8+2x_9+x_{10}=1$
  $0.000001 \leq x_i \leq 2 \quad (i=1,\ldots,10)$

  $c_1=-6.089 \quad c_2=-17.164 \quad c_3=-34.054$
  $c_4=-5.914 \quad c_5=-24.721 \quad c_6=-14.986$
  $c_7=-24.100 \quad c_8=-10.708 \quad c_9=-26.662$
  $c_{10}=-22.179$

۳. شرح عملکرد برنامه

به منظور حل مساله فوق، محیط شبیه سازی شامل یک دستگاه ٦٤ بیتی، Core i7 5500U 2.4 GHz با 8GB حافظه اصلی و Windows 10 Enterprise و نرم افزار MATLAB 2018b می‌باشد.

۳,۱. الگوریتم ژنتیک با کدینگ حقیقی

در این الگوریتم هر کروموزوم شامل مقدار متغیرهای توابع میباشدیعنی هر کروموزوم برداری است به اندازه متغییرهایی که تابع دارد. الگوریتم اصلی در فایل main می باشد و اجرا می شود.

پارامتر ها در این فایل عبارتند از:

۱. popsize: نشان دهنده میزان جمعیت میباشد که ما در تمامی اجرا ها میزان آن را برابر ۵۰ در نظر گرفته ایم.

۲.d: تعیین کننده تعداد متغیر های تابع میباشد .چون این الگوریتم باید بر روی ۳ تابع مختلف اجرا شود و هر سه تابع در تعداد متغیرهایشان باهم متفاوت میباشند یعنی تابع اول دو متغیره تابع دوم ۵ متغیره و تابع سوم ۱۰ متغیره میباشند مقداری که d هم میتواند بپذیرد با توجه به اینکه میخواهیم الگوریتم برای کدام تابع اجرا شود ۲ ، ۵ و ۱۰ می باشد.

۳.pc: نرخ آمیزش می باشد که تعیین کننده تعداد جمعیتی که قرار است در فرایند آمیزش برای تولید فرزند جدید شرکت کنند را تعیین میکند که این میزان با توجه به مساله ای که حل می کنیم متفاوت می باشد.

٤. pm : نرخ جهش می باشد. تعیین کننده تعداد جمعیتی که قرار است در فرایند جهش برای تولید فرزند جدیدشرکت کنند را تعیین میکند که این میزان با توجه به مساله ای که حل می کنیم متفاوت می باشد.

۵.maxgen : این متغیر ماکزیموم تعداد نسل هارا مشخص میکند که در حل تمامی مسایل آن را ۱۰۰۰ در نظر گرفته شده.

٦. landa: چون در الگوریتم ژنتیک نوشته شده در قسمت آمیزش آن از شیوه آمیزش محدب استفاده شده لازم بود که مقدار λ را مشخص کنیم که این مقدار برای هر مساله متفاوت می باشد.

۷.c & alpha & beta: پارامتر های کنترلی برای تابع جریمه می باشد.

الگوریتم ژنتیک نوشته شده شامل توابع زیر می باشد:

۱. تابع fitness

```
function F=fitness(p,d,t,c1,alpha,beta)
```

این تابع جمعیتی که قرار است مقدار تابع هدف برای آن مشخص شود را و هم چنین تعداد متغیر تابع(جهت تعیین اینکه کدام یک از توابع f1،f2 یا f3 باید اجرا شوند) و هم چنین شماره نسل هر جمعیت را به عنوان ورودی دریافت میکند.

خروجی تابع : میزان تابع هدف را به صورت حاصل جمع مقدار تابع و میزان جریمه برای ما باز میگرداند.

۲. تابع crossover

```
function crosspop=crossover(crosspop,pop,ncross,Landa,d,ngeneration,
```

```
c,alpha,beta)
```

ورودی های تابع :

۱. crosspop : ماتریسی که حاوی جمعیت حاصل از عمگر آمیزش بر روی ماتریس جمعیت ها می باشد.

۲. pop : ماتریس حاوی جمعیت اولیه می باشد.

۳. ncross : تعداد فرزندانی که از اعمال عملگر آمیزش بر روی ماتریس جمعیت pop به دست می آیند را مشخص می کند.

٤. Landa: چون در این الگوریتم از آمیزش محدب استفاده کرده ایم و جهت اعمال این آمیزش بر روی جمعیت لازم است از فرمول زیر استفاده کنیم

```
x1=Landa .* p1+(1-Landa).*p2;
x2=Landa .* p2+(1-Landa).*p1;
```

لذا میزان λ با این متغییر مشخص میکنیم.

٥. d : تعداد متغیر تابع (جهت تعیین اینکه کدام یک از توابع f1،f2 یا f3 باید اجرا شوند)

٦. ngeneration : شماره نسلی که داریم روی آن نسل عمل آمیزش را انجام میدهیم مشخص میکند . این شماره در تابع جریمه استفاده می شود.

خروجی تابع :

خروجی این تابع ماتریسی حاوی جمعیت حاصل از عملگر آمیزش می باشد.

۳. تابع mutation

```
function mutpop=mutation(mutpop,pop,d,nmut,ngeneration,maxgen,popsize,
c,alpha,beta)
```

ورودی های تابع :

۱. mutpop: ماتریسی که حاوی جمعیت حاصل از عمگر جهش بر روی ماتریس جمعیت ها می باشد.

۲. pop: ماتریس حاوی جمعیت اولیه می باشد

۳. d : تعداد متغیر تابع(جهت تعیین اینکه کدام یک ازتوابع f1،f2 یا f3 باید اجرا شوند)

٤. nmut : تعداد فرزندانی که از اعمال عملگر آمیزش بر روی ماتریس جمعیت pop به دست می آیند را مشخص می کند

٥. ngeneration :شماره نسلی که داریم روی آن نسل عمل آمیزش را انجام میدهیم مشخص میکند . این شماره در تابع جریمه استفاده می شود

٦. maxgen :این متغیر ماکزیموم تعداد نسل هارا مشخص میکند که در حل تمامی مسایل آن را ۱۰۰۰ در نظر گرفته ایم.

٧. popsize :نشان دهنده میزان جمعیت میباشد که ما در تمامی اجرا ها میزان آن را برابر ۵۰ در نظر گرفته ایم.

خروجی تابع :

خروجی این تابع ماتریسی حاوی جمعیت حاصل از عملگر جهش می باشد.

جهت عملگر جهش از جهش پویا (Dynamic mutation)استفاده شده.

چون در جهش پویا نیاز داشتیم $\Delta(t,y)$ را محاسبه کنیم در یک تابع جدا به نام delta این مقدار محاسبه شده و در اینجا از حاصل آن استفاده شده است.

٤. تابع delta

```
5. function dlt=delta(ngeneration,selectedGenforMut,maxX,minX,
random,maxGen)
```

ورودی تابع :

١. ngeneration : شماره نسل

٢. selectedGenforMut : ژنی که برای عمل جهش انتخاب شده

٣. maxX:باند بالای هر متغییر

٤. minX:باند پایین هر متغییر

٥. random : یک عدد رندم که با توجه به مقدار آن فرمولی که برای جهش از آن استفاده میشود مشخص می شود

٦. maxGen : این متغیر ماکزیموم تعداد نسل هارا مشخص میکند که در حل تمامی مسایل آن را ۱۰۰۰ در نظر گرفته ایم.

خروجی تابع :
مقدار تابع دلتا را برای ما باز میگرداند که در عملیات جهش از آن استفاده می شود.

توابع penaltyF1 و penaltyF2 و penaltyF3 مربوط به یافتن میزان جریمه ها در توابع f1و f2و f3 می باشد.

۳,۲. الگوریتم تکامل تفاضلی

الگوریتم اصلی در فایل main می باشد و اجرا می شود. در این تابع تعداد جمعیت اولیه (popsize)، تعداد متغیر d(متناسب با تابع انتخابی) و پارامتر های کنترلی تابع جریمه را وارد میکنیم تا خروجی، یعنی بهترین جواب (minimum) بدست بیاید.
برای جهش بردار آزمایشی را با جهش دادن دو بردار و یک بردار هدف که به صورت تصادفی انتخاب شده اند و همگی با هم متفاوتند به صورت زیر به دست می آوریم:

```
u(i,:)=pop(A,:,gen-1)+bt.*(pop(B,:,gen-1)-pop(C,:,gen-1));
```

ضریب Beta باید در ابتدای مسئله بزرگ انتخاب شود تا exploration بیشتری انجام شود و رفته رفته در نسل های آینده باید کاهش یابد تا در مناطق بهتر بیشتر تمرکز کند و استخراج انجام شود. ما این مسئله را با ضرایب متفاوت Beta بررسی کرده ایم.

برای آمیزش ابتدا یک متغیر تصادفی از بین درایه های بردار p انتخاب میکنیم تا فرزند ایجاد شده حداقل در یک متغیر با والدش متفاوت باشد، و آن را به مجموعه J اضافه میکنیم، سپس برای هر متغیر یک عدد رندوم تولید میکنیم و با pr (احتمال آمیزش) مقایسه میکنیم اگر کمتر از آن بود اندیس این متغیر به مجموعه J اصلی اضافه میشود. حال درایه های اندیس J از بردار آزمایشی را جایگزین بردار p نسل قبل میشود و بردار جدید newpop(i,J) بدست میاید.

```
newpop(i,J)=u(i,J);
```

حال fitnessاین بردار را با fitness بردار P نسل قبل مقایسه میکنیم اگر کوچیکتر از آن بود آن را جایگزینش میکنیم و Pنسل فعلی بدست می آید. Pr احتمال بکارگیری نقطه آمیزش است که هرچه بیشتر باشد تعداد نقاط بیشتری برای آمیزش انتخاب میشوند.

الگوریتم تکامل تفاضلی نوشته شده شامل توابع زیر می باشد:

۱. تابع fitness

```
function F=fitness(p,d,t,c,alpha,beta)
```

ورودی تابع:

P یک بردار جواب و درواقع یک solution و یا یک ذره میباشد که این بردار به تعداد d که همان متغیرهای مسئله است درایه دارد، t شماره نسل و Generation فعلی ما میباشد.

خروجی تابع:

خروجی این تابع F میباشد که fitness جمعیت نسل های بعد را محاسبه میکند.. برای محاسبه ی F ها قیود مسأله را با توابع جریمه روی مسأله اعمال کرده ایم.

۲. تابع penalty

```
function pen=penalty(p,d,t,c,alpha,beta)
```

برای متغیرهای infeasible جزء جریمه ای را از طریق تابع penalty و به دو روش که شرح آن ها در زیر ذکر شده است، محاسبه کرده و چون مسئله minimization است این تابع penalty را به fitness تابع اصلی اضافه می کنیم.

۴.

۵. شبیه سازی ها و نتایج

۵,۱. الگوریتم ژنتیک با کدینگ حقیقی

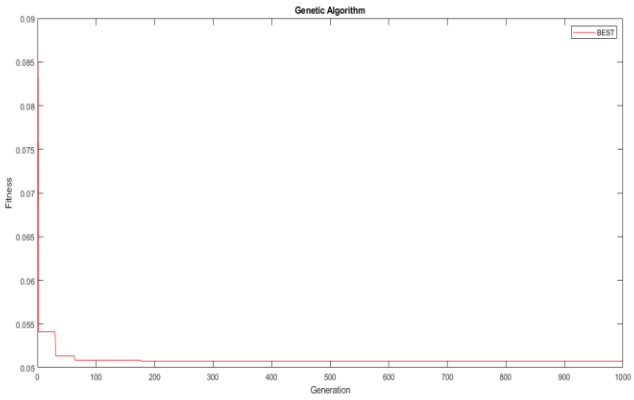
جدول ۱: نتایج حاصل از بهینه سازی توابع محک با استفاده از الگوریتم ژنتیک با کدینگ حقیقی

Function	Mean	Best	Worst	STD	Time (s)
F1(x)	0.0385	0.0048	0.0521	0.0182	0.0661
F2(x)	-3.1337e+04	-3.1699e+04	-3.0636e+04	347.3893	0.0454
F3(x)	-307.4668	-334.6148	-295.6932	13.9094	0.0740

- پارامتر های استفاده شده در تابع یک:

F1:

```
d=2;
pc=0.001;
pm=0.05;
landa=0.2;
alpha=1;
beta=3;
c=0.001;
```

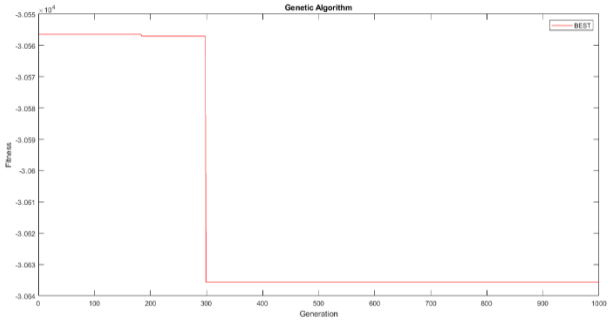


در اجراهای متوالی این تابع(f1) و با مقدار دهی متفاوت پارامترها، مقدار مینیمم تابع به صفر نزدیک میشد.

- پارامتر های استفاده شده در تابع دو:

F2:

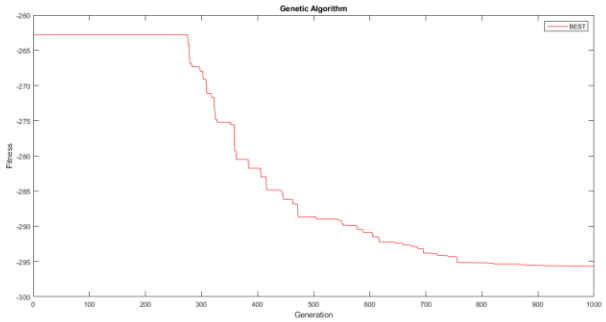
```
d=5;
pc=0.001;
pm=0.02;
landa=0.9;
alpha=0;
beta=0.9;
c=9;
```



- پارامتر های استفاده شده در تابع سه:

F3:

```
d=10;
pc=0.009;
pm=0.03;
landa=0.8;
alpha=3;
beta=0.0002;
c=0.099;
```



در اجرای این تابع مقادیر به دست آمده از ۲۰- تا ۴۲۰- متغییر بودند در نهایت با توجه به تنظیم پارامترها مقدار مینیمم تقریبا به ۳۰۰- نزدیک بود.

۵,۲. الگوریتم تکامل تفاضلی

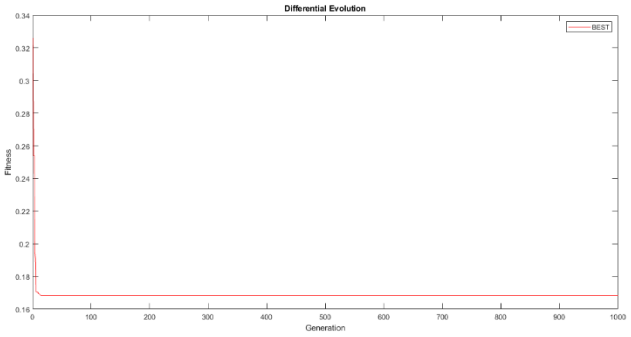
جدول ۲: نتایج حاصل از بهینه سازی توابع محک با استفاده از الگوریتم تکامل تفاضلی

Function	Mean	Best	Worst	STD	Time (s)
F1(x)	0.1683	0.1683	0.1683	2.0669e-06	1.6529
F2(x)	-3.0981e+04	-3.1170e+04	-3.0761e+04	122.0860	1.7736
F3(x)	-172.9960	-173.7422	-172.0041	0.6147	2.1483

- پارامتر های استفاده شده در تابع یک:

F1:

```
d=2;
bt=0.5;
pr=0.5;
alpha=0.001;
beta=2;
c=2;
```

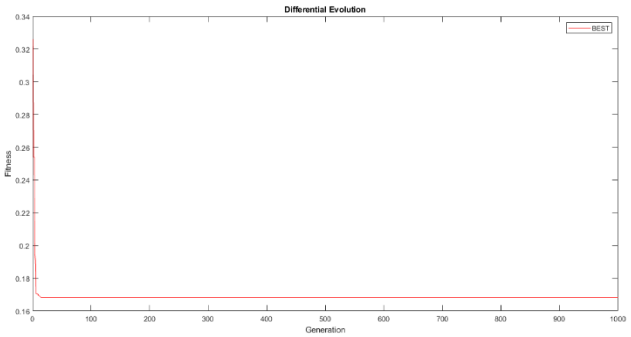


در این الگوریتم پارامترهای تابع penalty را بدین صورت در نظر گرفته شده، alpha را خیلی کوچک در نظر گرفته . هرچه پارامتر beta راکوچک و عددی نزدیک صفردر نظر بگیریم تغییرات جواب شدیدتر می شود ولی جواب بهینه تا اندازه ای بدتر می شود بنابراین پارامتر beta را برابر با ۲گرفته ایم.

- پارامتر های استفاده شده در تابع دو:

F2:

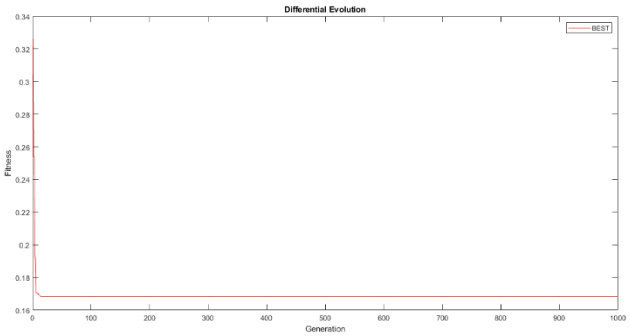
```
d=5;
bt=0.1;
pr=0.5;
alpha=0.01;
beta=2;
c=2;
```



- پارامتر های استفاده شده در تابع سه:

F3:

```
d=10;
bt=0.001;
pr=0.2;
alpha=0.01;
beta=2;
c=2;
```



نتیجه گیری کلی

در مقایسه الگوریتم تکامل تفاضلی با ژنتیک باید گفت با توجه به مقادیر به دست آمده برای std ها دقت تکامل تفاضلی از ژنتیک بیشتر است زیرا در الگوریتم ژنتیک بر اساس شایستگی والدین به آنها شانس انتخاب شدن میدهد در صورتی که در الگوریتم تفاضلی تمامی والدین شانس یکسان برای انتخاب شدن دارند . ولی باتوجه به نتایج به دست آمده زمان اجرای تکامل تفاضلی از ژنتیک بالاتر است.