به نام خدا

گزارش تمرین چهارم مبانی هوش

الگوریتم ژنتیک

غزل زمانی نژاد، ۹۷۵۲۲۱۶۶

۱) در این سوال باید مسئله فروشنده دوره گرد به کمک الگوریتم ژنتیک حل شود. برای این کار ابتدا کلاس TSP را تعریف می کنیم که در constructor آن تعداد شهرها، فاصله میان همه شهرها، تعداد کروموزوم در هر نسل (همان population) و تعداد نسل ها وجود دارد. به بررسی سایر متدهای این کلاس می پردازیم:

Generate: این تابع ۱ کروموزوم ایجاد می کند. برای این کار به کمک random.shuffle یک رشته تصادفی از شهرها تولید می کنیم. و چون در این سوال فروشنده در پایان به شهر اولیه خود برمیگردد، عنصر آخر رشته را همان عنصر اول قرار می دهیم.

Fitness: باید بر اساس کروموزوم ورودی تابع و همچنین فاصله میان شهرها، طول کل مسیر را محاسبه کنیم. در این سوال هرچه مقدار fitness کمتر باشد، رشته تولید شده پاسخ بهتری خواهد بود. پس برای این کار، فاصله میان شهرهای ورودی تابع را در لیست total ذخیره می کنیم. اگر در آن ۱- موجود باشد یعنی مسیری وجود ندارد (مسیر تولیدشده اشتباه است) پس به جای آن بی نهایت برمیگردانیم. در غیر این صورت، طول کل مسیر را ریترن میکنیم.

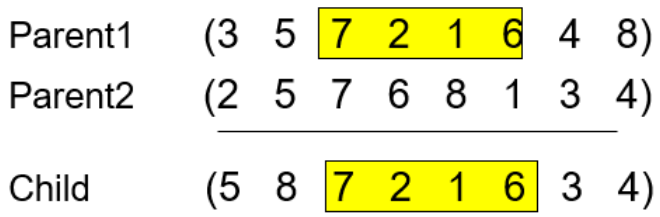
Reproduction: برای پیاده سازی آن از روش Elititst استفاده می کنیم. توضیح روش:

•“Elitist strategy” ensures that individual with highest fitness is copied into next generation (most GAs use this)

بدین منظور، باید از مجموعه ای از پاسخ ها، ایندکس پاسخ با بهترین فیتنس ها را پیدا کنیم. از تابع آماده np.argpartion استفاده می کنیم تا k (تعداد پرنت هایی که میخواهیم از آنها نسل جدید تولید کنیم) عنصر با کمترین مقدار فیتنس را محاسبه کند. سپس آن ایندکس ها را ریترن می کنیم.

Crossover: برای crossover از order1 استفاده کرده ایم. در این روش، دو ایندکس رندوم از لیست انتخاب می کنیم. مقدار کوچکتر را در a و مقدار بزرگتر را در b میریزیم. سپس از ایندکس a تا

b پرنت اول را به اول child اضافه میکنیم. سپس روی پرنت دوم iterate می کنیم و هر مقداری که داخل child نبود به c اضافه می کنیم. و با استفاده از متد extend آن را به child اضافه می کنیم.
مثال آن در شکل زیر مشاهده می شود.



در پایان اگر عنصر آخر لیست با عنصر اول برابر نباشد آن را تصحیح می کنیم.
Mutation: برای جهش دادن یک کروموزوم، دو ایندکس را به صورت رندوم انتخاب میکنیم و آنها را باهم جا به جا میکنیم.

توضیح کلیت کد: ابتدا در یک ماتریس ۷*۷، فاصله ها را ذخیره میکنیم. اگر بین دو شهر مسیر وجود نداشته باشد، خانه متناظر با آن را ۱- میگذاریم. قطر اصلی ماتریس را با ۰ پر میکنیم. سپس نمونه ای از کلاس TSP با تعداد کروموزوم ۱۰ و تعداد نسل ۲۰۰ میسازیم. و متد GA را فراخوانی میکنیم. در این متد ابتدا به تعداد population با استفاده از متد Generat، کروموزوم تولید میکنیم. بعد در یک لوپ به تعداد نسل ها مرتبا کار زیر را انجام میدهیم:
تعداد پرنت ها را ۴ درنظر گرفتیم. با استفاده از متد Reproduction، ایندکس بهترین پاسخ ها را می یابیم و آنها را به لیست parents اضافه میکنیم. روی تمام پرنت ها iterate میکنیم و با همه جایگشت های ممکن پرنت ها با تابع Crossover کروموزوم جدید تولید میکنیم. سپس pop را خالی میکنیم و پرنت ها را به آن لیست اضافه میکنیم. سپس با تابع mutation روی فرزندها جهش اعمال میکنیم و به لیست pop اضافه میکنیم. از این لیست برای نسل بعد استفاده میکنیم.
نتیجه در شکل زیر مشاهده میشود.

```
best solution, iteration 0 : [6, 2, 5, 1, 7, 4, 3, 6]
length inf
best solution, iteration 40 : [7, 6, 4, 2, 1, 3, 5, 7]
length 63
best solution, iteration 80 : [7, 6, 4, 2, 1, 3, 5, 7]
length 63
best solution, iteration 120 : [5, 7, 6, 4, 2, 1, 3, 5]
length 63
best solution, iteration 160 : [6, 4, 2, 1, 3, 5, 7, 6]
length 63
best solution is: [5, 7, 6, 4, 2, 1, 3, 5]
length: 63
```

۲) در این سوال میخواهیم ریشه تابع را به کمک الگوریتم ژنتیک بیابیم. برای این کار ابتدا کلاس Equation را تعریف می کنیم که در constructor آن طول هر کروموزم، تعداد کروموزوم در هر نسل (همان population) و تعداد نسل ها وجود دارد. به بررسی سایر متدهای این کلاس می پردازیم:

Value: این متد مقدار ورودی را میگیرد و آن را به تابع سوال میدهد و خروجی آن را برمیگرداند.

Generate: این تابع ۱ کروموزوم رندوم ایجاد می کند. در این رشته ۶ بیتی بیت صفرم نشان دهنده علامت عدد است (اگر بیت صفر زوج باشد، عدد مثبت و در غیر این صورت عدد منفی است.) بیت اول نشان دهنده مقدار صحیح جواب است. بیت دوم تا پنجم نشان دهنده قسمت اعشاری عدد است.

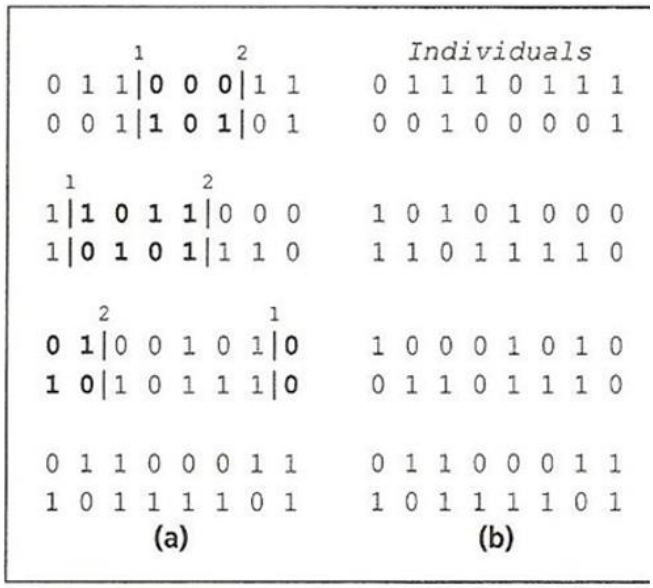
ConvertListToNum: این متد یک استرینگ از کروموزوم میگیرد و آن را به عدد تبدیل میکند. در ابتدا بیت اول را به int، parse میکند. بعد از بیت دوم تا آخر را به float، parse میکند. دو مقدار int و float را باهم جمع میکنیم، در صورت فرد بودن بیت صفرم آن را در ۱- ضرب میکنیم. عدد را ریترن میکنیم.

Fitness: در این سوال هرچه مقدار fitness کمتر باشد رشته تولید شده پاسخ بهتری خواهد بود. پس برای این کار، قدرمطلق مقدار تابع را فیتنس درنظر میگیریم.

Reproduction: برای پیاده سازی آن از روش Elititst استفاده می کنیم.

بدین منظور، باید از مجموعه ای از پاسخ ها، ایندکس پاسخ با بهترین فیتنس ها را پیدا کنیم. از تابع آماده np.argpartion استفاده می کنیم تا k (تعداد پرنت هایی که میخواهیم از آنها نسل جدید تولید کنیم) عنصر با کمترین مقدار فیتنس را محاسبه کند. سپس آن ایندکس ها را ریترن می کنیم.

Crossover: برای crossover از two point استفاده کرده ایم. در این روش، دو ایندکس رندوم از لیست انتخاب می کنیم. و مقدار بین دو ایندکس را با هم جا به جا میکنیم تا دو فرزند جدید بسازیم.
هر دو فرزند جدید را ریترن میکنیم.
مثال آن در شکل زیر مشاهده می شود.



Mutation: برای جهش دادن یک کروموزوم، دو ایندکس را به صورت رندوم انتخاب میکنیم و آنها را باهم جا به جا میکنیم.

توضیح کلیت کد: نمونه ای از کلاس Equation با تعداد کروموزوم ۵۰۰ و تعداد نسل ۲۰۰۰ میسازیم. و متد GA را فراخوانی میکنیم. در این متد ابتدا به تعداد population با استفاده از متد Generat، کروموزوم تولید میکنیم. بعد در یک لوپ به تعداد نسل ها مرتبا کار زیر را انجام میدهیم:
تعداد پرنت ها را ۴ درنظر گرفتیم. با استفاده از متد Reproduction، ایندکس بهترین پاسخ ها را می یابیم و آنها را به لیست parents اضافه میکنیم. روی تمام پرنت ها iterate میکنیم و با تابع Crossover کروموزوم جدید تولید میکنیم. سپس pop را خالی میکنیم و پرنت ها را به آن لیست اضافه میکنیم. سپس با تابع mutation روی فرزندها جهش اعمال میکنیم و به لیست pop اضافه میکنیم. از این لیست برای نسل بعد استفاده میکنیم.
نتیجه در شکل زیر مشاهده میشود.

```
best solution is: 4.884
f(solution) =  0.00010777905481518246
```