گزارش سوال سودوکو

محمد فرهی شماره دانشجویی: ۸۱۰۱۹۸۴۵۱

بخش اول :

در این بخش، تعریفی که برای ژن در نظر گرفته شده است، یک سطر از جدول سودوکو است. هر سطر که اعداد ۱ تا ۹، بدون تکرار و به صورت رندوم در آن قرار دارد، به عنوان ژن در نظر گرفته می شود و نتیجتا کروموزوم یک آرایه ۹ تایی از این سطر ها خواهد بود(Condidate.solution). در حقیقت سطر های جدول، ژن در نظر گرفته می شوند. کلاس Condidate به همراه متد های مورد نیاز، برای این کار پیاده سازی شده است

بخش دوم:

در این بخش، برای تولید جمعیت اولیه رندم ، ابتدا برای هر خانه از جدول با توجه به جدول اولیه ورودی، مقادیری که تناقضی با مقادیر ایستا(جزء جدول اولیه) ندارند را پیدا می کنیم.(Population.possible_values) سپس برای هر خانه از سطر(ژن) به طور رندم از مقادیر مقدور انتخاب کرده و خانه های سطر را مقدار دهی می کنیم و به این نکته توجه داریم که مقادیر سطر تکراری نباشند. برای هر ۹ سطر این کار را انجام می دهیم و به این ترتیب کروموزوم رندم تولید می شود(Population._generate_random_condidate). جمعیت اولیه در Population.generate_initial ساخته می شود.

بخش سوم:

در این بخش، معیاری که برای fitness در نظر گرفته می شود،تعداد عدد های یکتا در هر ستون و در هر بلاک(مربع های ۳*۳) است. حداکثر این مقدار ۱۶۲ است(در کل ۱۸ بلاک و ستون داریم که تعداد عدد های یکتا در هر یک حداکثر ۹ است:۹*۱۸ = ۱۶۲) و اگر کاندیدا ای با این فیتنس پیدا شود، جواب است. این تابع در Condidate.calculate_fitness پیاده سازی شده و در هر جایی که

کروموزوم آن کاندیدا تغییر کند (مثلا در mutation) این تابع صدا زده می شود.

بخش چهارم:

کراس اور (apply_crossover): در این تابع (apply_crossover) بر روی دو parent که پاس داده شده اند با احتمال Pc، کراس اور uniform اجرا می کنیم. برای هر دو سطر(ژن)با احتمال ۱/۲ آنها را جابه جا می کنیم. اگر حداقل یک جابه جایی صورت بگیرد، پرچی را یه عنوان تولید دو کروموزم جدید از تابع برگردانده می شود.

میوتیشن(Condidate.apply_mutation): در این متد، بر روی هر سطر با احتمال Pm عملیات میوتیشن انجام می شود. از میان تمام دو خانه های موجود در سطر، اولین دو خانه ای که جابه جا کردن مقدار آن ها با هم، تناقضی در بلاک یا ستون متناظر آن دو خانه نگذارد، مقادیر آن دو خانه با هم عوض می شوند و میوتیشن تمام میشود. اگر میوتیشن حداقل در یک سطر صورت پذیرد، پرچمی برای اعلام موفق بودن میوتیشن برگردانده می شود. به دلیل پرهزینه بودن میوتیشن در هر سطر،پارامتری برای مشخص کردن تعداد سطر هایی که میوتیشن بر رویشان انجام می شود به متد پاس داده میشود.(gen_mutate_count)

بخش پنجم:

توضیح کلی پیاده سازی:

ابتدا شیء population ساخته می شود. سپس به کاندید های اولیه ساخته می شوند(genetate_initial) و متد sort که وظیفه مرتب کردن کاندیداها بر اساس فیتنس آن ها را بر عهده دارد، صدا زده می شود سپس در هر سیکل:

۱- وجود جواب در میاد کاندیدا ها چک می شود.

۲- درصدی از کاندیداهای برتر به طور مستقیم برای نسل بعد انتخاب می شوند. سپس برای تکمیل جمعیت نسل بعد، از میان جمعیت برای استخر جفت گیری نمونه برداری می شود. برای نمونه برداری از روش binary tournament selection استفاده می شود.

۳- بر روی کاندیدا های منتخب عملیات جفت گیری صورت می گیرد. اگر عملیات کراس اور موفقیت آمیز نباشد و parent ها از درصد برتر جامعه بوده باشند(elite) برای نگهداری از diversity جامعه به جای آنها از کاندیدا هایی با فیتنس کمتر جامعه فعلی وارد جامعه بعدی می شود(چون کاندیدای elite یکبار مستقیم وارد شده). بعد از آن Pc برای نسل بعد تعیین می شود
۴- بر روی کاندیدا هایی که از نتیجه کراس اور بیرون آمده اند، میوتیشن انجام می گیرد و بعد از آن مقدار Pm برای سیکل بعد مشخص می شود.
۵ - در نهایت مکانیزمی برای رهایی از گیر افتادن در لوکال مینیمم انجام می گیرد.

بخش ششم:

پاسخ به سوالات:

۱- برای انتخاب کاندیداهای mating pool از روش binary tournament selection استفاده شده است. در این روش دو کاندیدا به صورت رندم از جمعیت فعلی انتخاب می شود. سپس با احتمال SELECTION_RATE کاندیدا با فیتنس بهتر،برای شرکت در کراس اور انتخاب می شود. و با احتمال مکمل آن هم کاندیدا با فیتنس کمتر. با این روش نسبت به روش roulette wheel زودتر به جواب اپتیمم می رسیم. همچنین در این روش diversity جمعیت بهتر کنترل می شود. چون در روش دیگر، فیتنس بالا شانس انتخاب برای در کراس اور و نقش داشتن در نسل بعد را افزایش می دهد.

۲- چون بسیار دقیق است!!

۳- با تغییر دادن هایپر پارامتر ها و تست فراوان،به نظر می رسد crossover بیشتر در نزدیک کردن الگوریتم سرچ و جمعیت به نقطه اپتیمم(که ممکن است محلی باشد) تاثیر دارد. اما از طرفی mutation بیشتر در گسترش دادن دامنه سرچ و جلوگیری از سو گرفتن جمعیت به یک نقطه و مقدار خاص تاثیر دارد. در نتیجه اگر در سیکل های ایتدایی بهتر است crossover rate مقدار (به نسبت خود) کمتری داشته باشد و mutation rate مقدار بیشتری داشته باشد.

اما رفته رفته زمانی که به نقطه اپتیمم نزدیک می شویم، مقدار crossover rate را بیشتر کنیم و از مقدار mutation rate کم کنیم.
در این پیاده سازی برای هر کدام از PC , PM یک مین و مکس در نظر گرفته شده است:
PC در ابتدا یک مقدار اولی دارد و برای تغییر دادن PC این گونه عمل می کنیم که در هر سیکل اگر نسبت کراس اور های موفق انجام شده از آنچه در PC برایش تعیین کرده بودیم کم شده کمتر بود، برای سیکل بعد PC را اندکی افزایش می دهیم تا در نهایت به یک مقدار مکسی در سیکل های بعدی برسیم.
PM نیز یک مقدار اولیه دارد. سپس در هر سیکل، از بین میوتیشن های موفق انجام شده اگر نسبت میوتیشن هایی که باعث بهتر شده آن کاندیدا شده است(بهتر کردن فیتنس) کمتر از مقدار مشخصی باشد(0.3) شروع به کم کردن PM می کنیم تا به یک مقدار مینیمم ای در سیکل های مختلفی برسیم. چون کم بودن این نسبت نشان می دهد که دیگر میوتیشن موثر عمل نمی کند. این می تواند به خاطر این باشد که بیشتر جمعیت و سرچ به یک مقدار اپتیمم ای نزدیک شده است. پی کم کردن PM تصمیم خوبی است.
۴- دلیل این اتفاق این است که عملیات سرچ به یک مقدار اپتیمال محلی رسیده است و تغییر وضعیت کاندیداها جمعیت را به موقعیت بدتری نسبت به وضعیتی که الان قرار دارند می برد.
مشکل این گیر کردن این است که یه جواب اصلی نمی رسیم.
برای حل می توان ای گونه عمل کرد که تعداد سیکل هایی را که این شرایط را داریم را شمرد و اگر از مقدار معینی (RESET_COUNT) این تعداد بیشتر شد، همه چیز را از نو شروع کرد. یعنی تولید جمعیت اولیه جدید و ریست کردن پارامتر های داینامیک الگوریتم(شماره ۵ بخش پنجم)
اجرای تست ها:

```
mohammad@mohammad-X550VX:~/AI/2/work$ python3 genetic.py < Test1.txt
************************************
answer found!!
fitness(Number of unique numbers in cols and blocks (max=162)): 162
8 2 6 9 3 5 1 4 7
4 1 7 6 8 2 9 5 3
9 5 3 1 7 4 8 2 6
7 9 4 8 2 1 6 3 5
5 6 8 3 4 7 2 9 1
1 3 2 5 6 9 4 7 8
3 4 1 2 5 8 7 6 9
2 8 5 7 9 6 3 1 4
6 7 9 4 1 3 5 8 2
exe time(s): 3.2264349460601807
mohammad@mohammad-X550VX:~/AI/2/work$ python3 genetic.py < Test2.txt
************************************
answer found!!
fitness(Number of unique numbers in cols and blocks (max=162)): 162
9 6 8 2 5 3 4 7 1
4 7 5 1 9 6 3 8 2
3 1 2 4 8 7 6 9 5
2 5 1 9 4 8 7 6 3
7 9 3 6 2 5 8 1 4
8 4 6 3 7 1 2 5 9
1 8 7 5 3 2 9 4 6
6 2 9 8 1 4 5 3 7
5 3 4 7 6 9 1 2 8
exe time(s): 2.9496567249298096
mohammad@mohammad-X550VX:~/AI/2/work$
```