دانشکده مهندسی
کامپیوتر و فناوری اطلاعات

گزارش پروژه
هوش مصنوعی

محمدرضا اخگری زیری

۹۶۳۱۰۰۱

زمستان ۹۸

مکعب روبیک

مدلسازی:

برای حل این مساله، ابتدا اقدام به مدل کردن مساله میکنیم. یک مکعب روبیک را به صورت یک آرایه دوبعدی در نظر میگیریم. که در هر خانه این آرایه اعدادی که نمایشگر رنگ آن خانه است قرار گرفته اند.

(برای سادگی در خواندن کد، برای این رنگها متغیر تعریف شده است.)

اعمالی که بر روی روبیک میتوان انجام داد، چرخش هر کدام از جهات به صورت ساعتگرد یا پادساعتگرد است.

مجموعه اعمال را در کد به شکل زیر نمایش دادیم:

```
ur = 0
ul = 1
lr = 2
ll = 3
fr = 4
fl = 5
rr = 6
rl = 7
dr = 8
dl = 9
br = 10
bl = 11
```

که تمامی این اعمال در action در کلاس problem قرار دارد.

با انجام این action ها بر روی روبیک به حالت جدیدی میرسیم که باید با تابع test_goal در کلاس یاد شده، تشخیص داد که آیا به هدف رسیده ایم یا خیر. لازم به ذکر است که اعمال با ۶ حرکت نیز قابل پیاده سازی بود که به دلیل خواسته سوال از ۱۲ حرکت استفاده شده است.

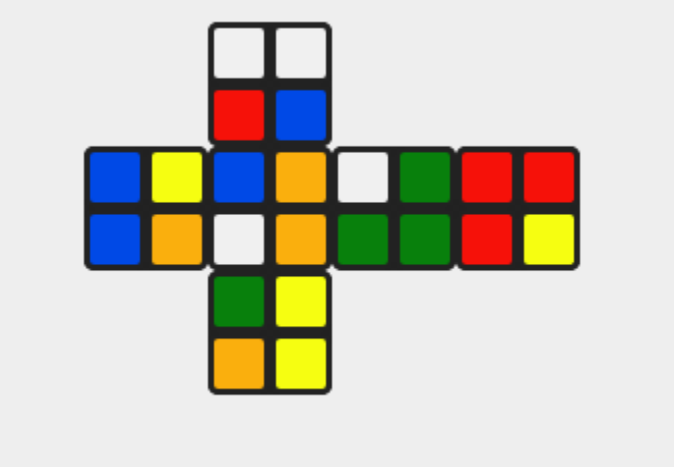
روش های حل این سوال به شرح زیر میباشد:

۱. جستجوی عمق اول با افزایش تدریجی عمق IDS:

در این مساله، هدف پیاده سازی الگوریتم dfs برای عمق های متفاوت است. یعنی در ابتدای الگوریتم عمقی را مدنظر قرار میدهیم و برای آن الگوریتم dfs را پیاده سازی میکنیم. اگر در عمق مورد نظر در یکی از شاخه ها به حالت هدف رسیدیم، الگوریتم عمق را باز میگرداند، در غیر اینصورت حداکثر عمق را افزایش میدهد و دوباره الگوریتم را از سر میگیرد. این روش باعث میشود تا ما به جواب بهینه برسیم، بدین صورت که در dfs اگر جواب در عمق بالاتری (نزدیک به ریشه) بود، امکان داشت به حالت هدف دیگری برسیم در عمق پایینتر. ولی در این روش اطمینان داریم که جواب پیدا شده بهینه است، در غیر اینصورت در حالت قبل (عمق قبلی) به آن جواب دست می‌یافتیم.

نتایج حاصل از کد برای این الگوریتم:

(متاسفانه به دلیل زیاد بودن حالت، مجبور به حل دستی مکعب بودم و چند مرحله جلوتر بردمش، تا به صورت زیر دربیاید)



نتایج به شرح زیر شد:

```
finial depth is : 3
Produced Nodes is: 556
Expanded Nodes is: 556
Max memory is: 4
```

۲. الگوریتم UCS:

این الگوریتم همانند bfs است، با این تفاوت که در هنگام انتخاب برای بسط آموزن هدف را اجرا میکنیم و در هر مرحله گره ای را انتخاب میکند که فاصله اش از ریشه کمترین باشد.
باز به دلیل طول کشیدن مراحل از حالت سوال قبل استفاده میکنیم.

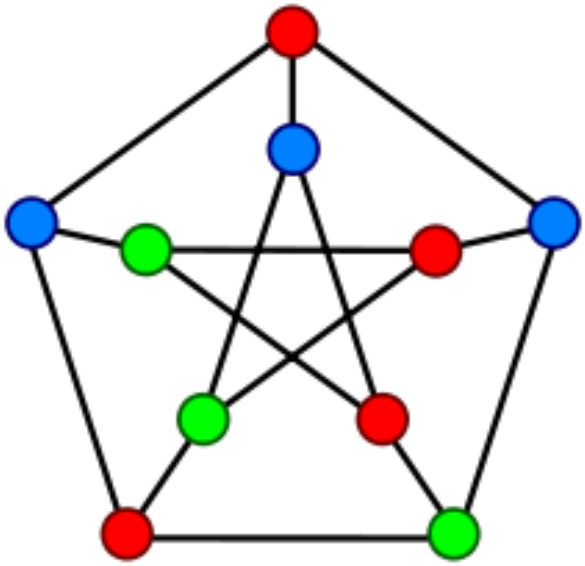
```
finial depth is : 3
Produced Nodes is: 8379
Expanded Nodes is: 943
Max memory is: 9322
```

۳. الگوریتم دوجهته:

در این روش از سمت حالت اولیه و از سمت حالت هدف به طور همزمان با استفاده از جستج وی اول سطح به دنبال راهحل میگردیم .به علت بهینه بودن الگوریتم اول سطح، هر گاه این دو جستجو به یکدیگر رسیدند، راه حل بهینه پیداشده است.

نتایج به شرح زیرند:
متاسفانه تو لوپ افتاد ☺

رنگ‌آمیزی گراف



در این سوال برای مدلسازی به شیوه زیر عمل کردیم و هر شهر را به یک عدد اختصاص دادیم و گرافی با آن میسازیم.

```
# list of cities
west_azarbaijan = 0
east_azarbaijan = 1
ardabil= 2
kurdestan = 3
zanjan = 4
gilan = 5
Kermanshah = 6
Hamedan = 7
Qazvin = 8
Mazandaran = 9
Golestan = 10
Ilam = 11
Lorestan = 12
Markazi = 13
Alborz = 14
Tehran = 15
Khuzestan = 16
ChaharmahalAndBakhriari = 17
Isfahan = 18
Qom = 19
Semnan = 20
NorthKhorasan = 21
RazaviKhorasan = 22
SouthKhorasan = 23
Yazd = 24
Kerman = 25
Fars = 26
Bushehr = 27
Hormozgan = 28
SistanAndBaluchestan = 29
KohgiluyehAndBoyer = 30
```

در فایل Graph_Creator ساختن گراف انجام شده است.

۱. ژنتیک:

در روش ژنتیک باید به تعداد population_size تعدادی کرومزم با حالت تصادفی ایجاد میکنیم. (در تابع generate_random_initial_population این اتفاق می‌افتد). در هنگام ساختن این کرومزم ها باید تابع برازش[۱] آنها را با فرمول درون صورت سوال مقدار دهی کنیم.

فرمول برازش به شرح زیر است:

تابع δ را به صورت زیر تعریف می‌کنیم :

$$for\ all\ i, j \in \mathrm{E}, \delta(i, j) = 1\ if\ c(i) \neq c(j)\ and\ \delta(i, j) = 0\ if\ c(i) = c(j)$$

و با استفاده از این تابع، fitness function مورد نظر ما به صورت زیر تعریف می‌شود :

$$F(C(G)) = \frac{\Sigma_{i, j \in E}\ \delta(i, j)}{m}$$

سپس اقدام به انجام الگوریتم میکنیم.

در ابتدای الگوریتم به تعداد tornumentSize باید عملیات انتخاب نخبه اتفاق بیفتد. و بهترین ها انتخاب کنیم. بعد باید cross over و جهش را روی این نتایج انجام دهیم و به جمعیت بیفزایم.

این کارها را تا انتهای مراحل ژنتیک یا هنگامی که به بهترین نتیجه برسیم ادامه میدهیم.

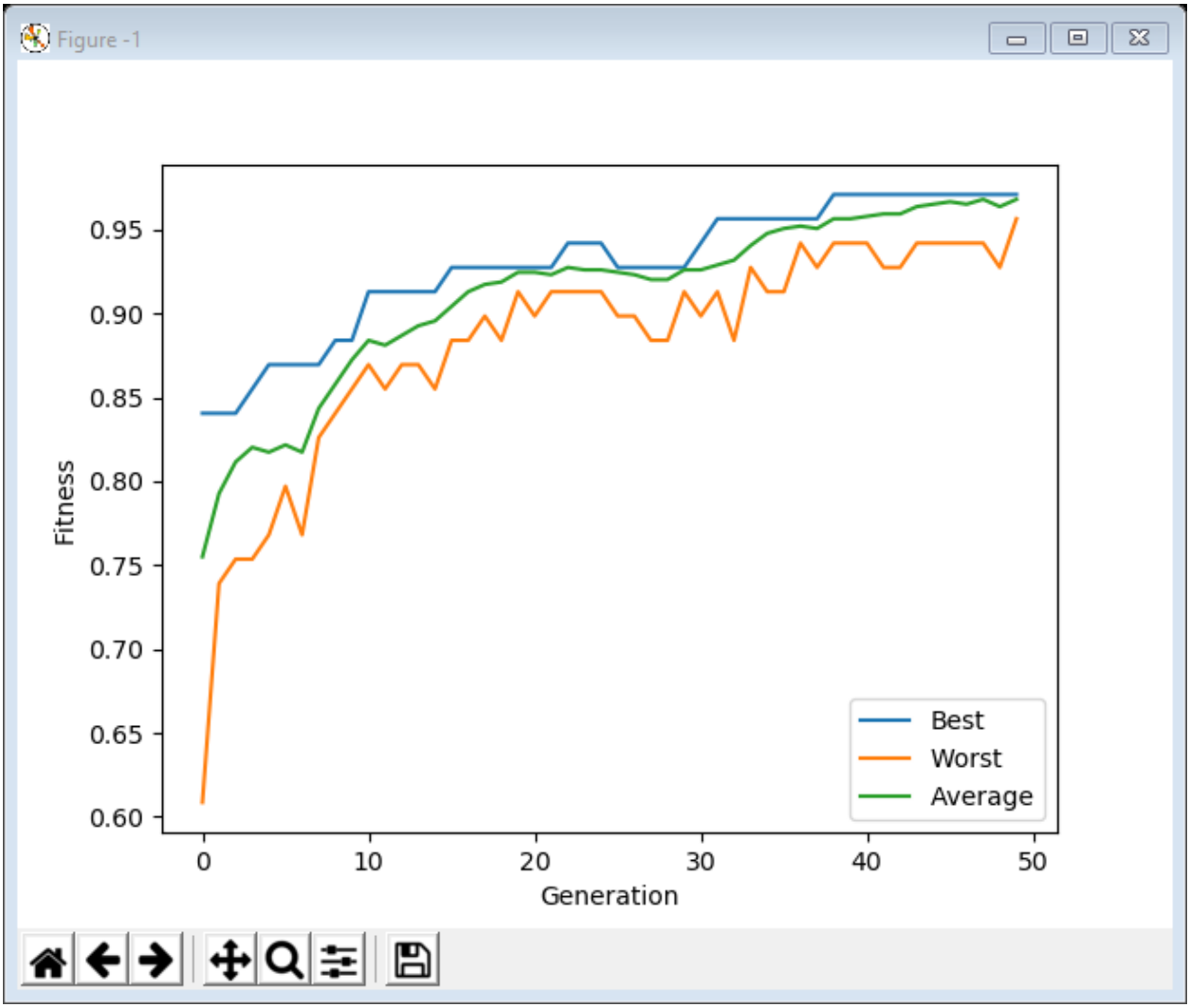
برای ورودی‌های سوال نتایج به شرح زیر میباشد:

[۱] fitness

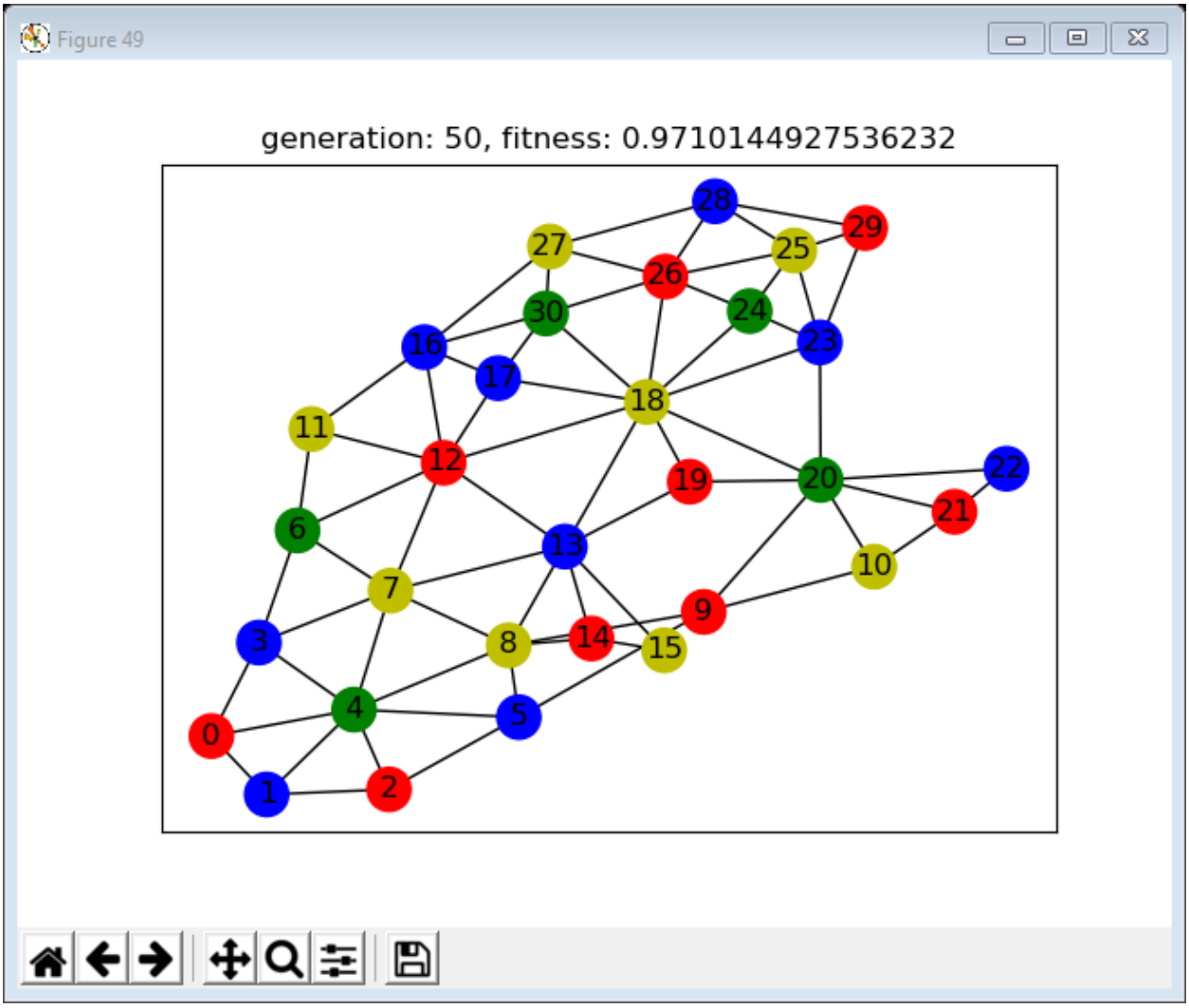
حالت اول:

```
population_size = 10 # number of initial population
muniationRate = 0.01
n_generations = 50 #  times
tornumentSize = 2  # In each generation update a certain number of ran
dom parents is kept
```

در این حالت به ۵۰ امین مرحله میرسیم ولی به برازش ۱ نمیرسیم. نمودارها به شکل زیر میباشند.



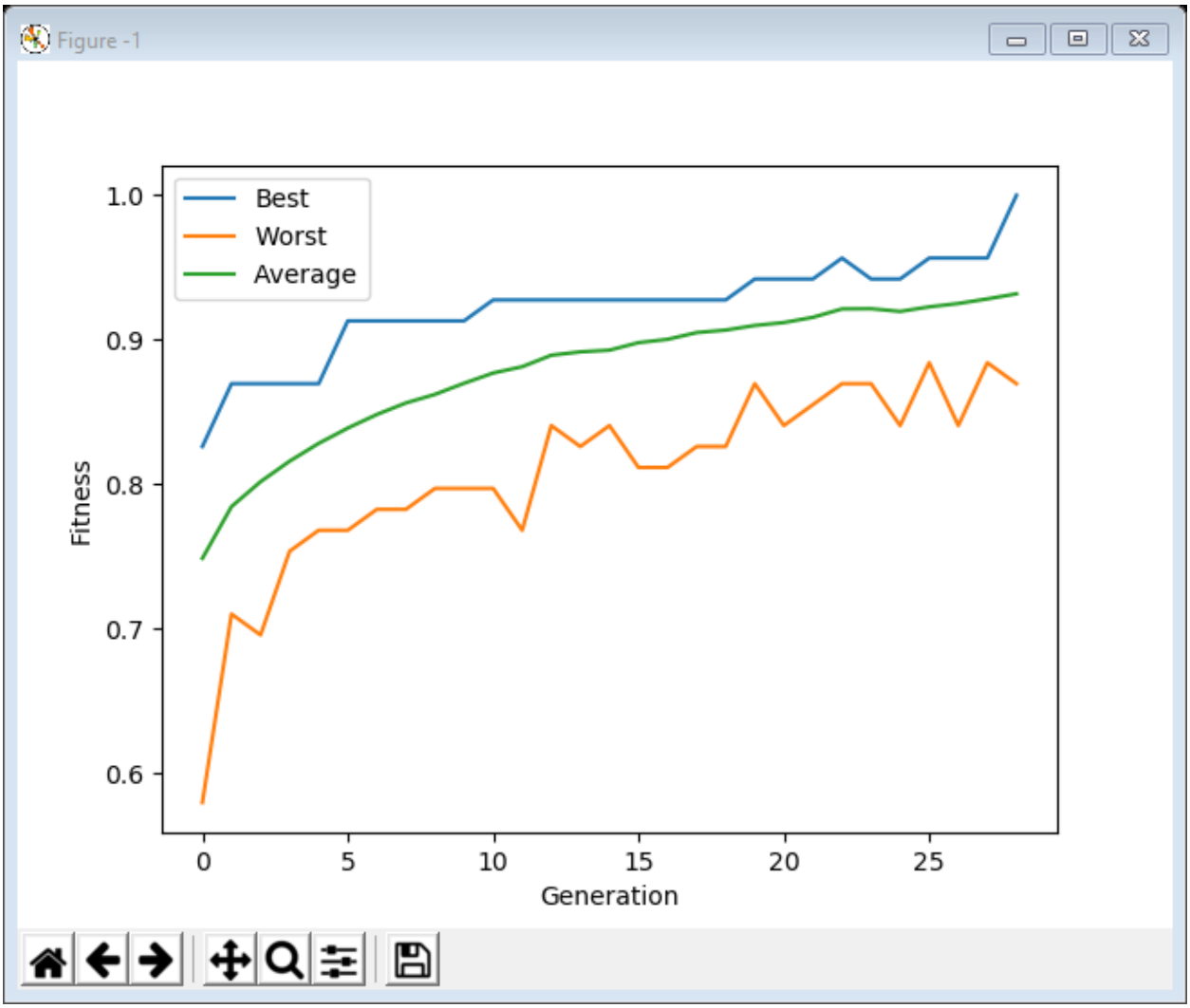
در انتها نقشه به شکل زیر است:



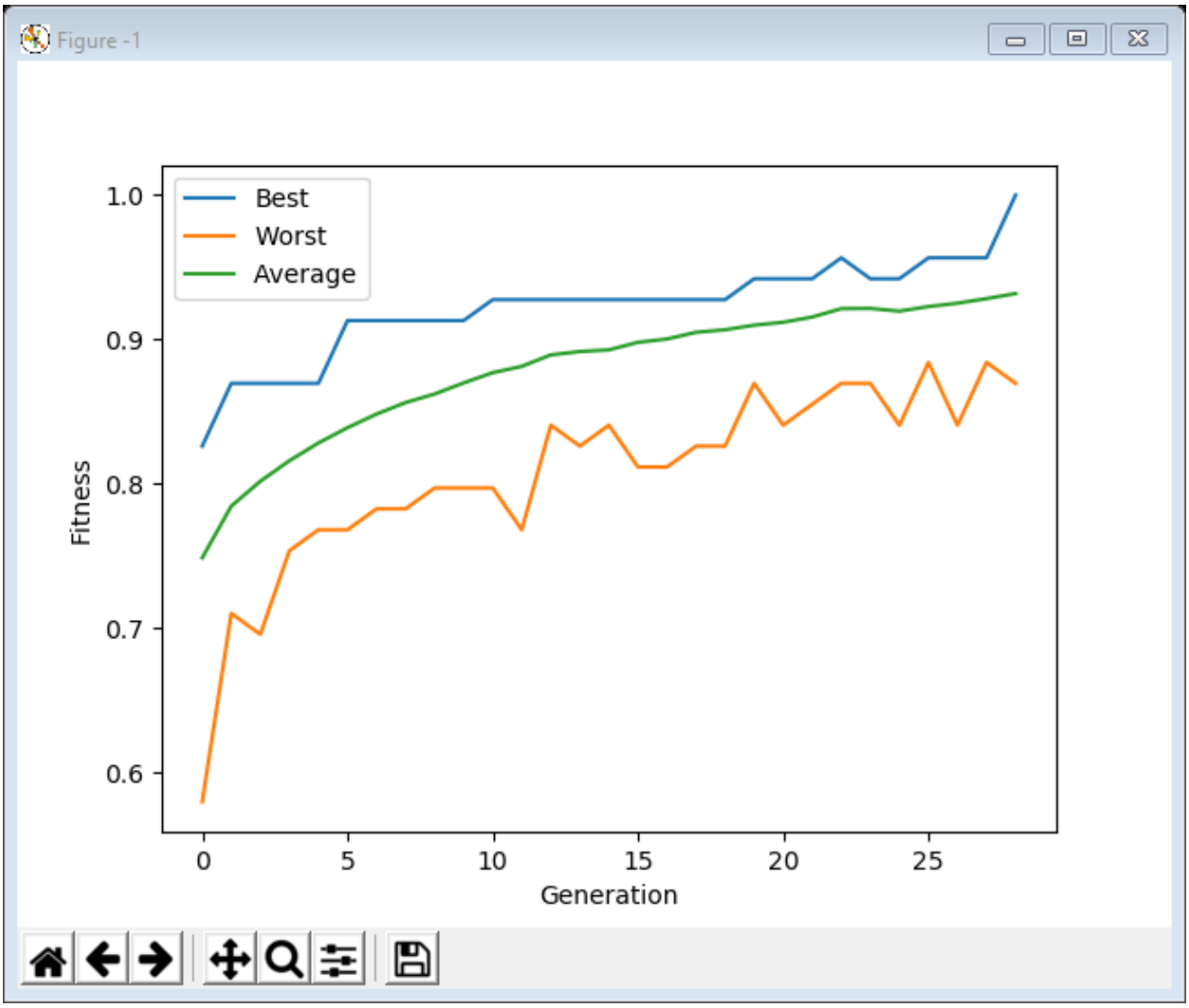
حالت دوم:

```
population_size = 100 # number of initial population
muniationRate = 0.01
n_generations = 50 #  times
tornumentSize = 2  # In each generation update a certain number of ran
dom parents is kept
```

در این حالت در ۲۹ امین مرحله به نتیجه میرسیم.



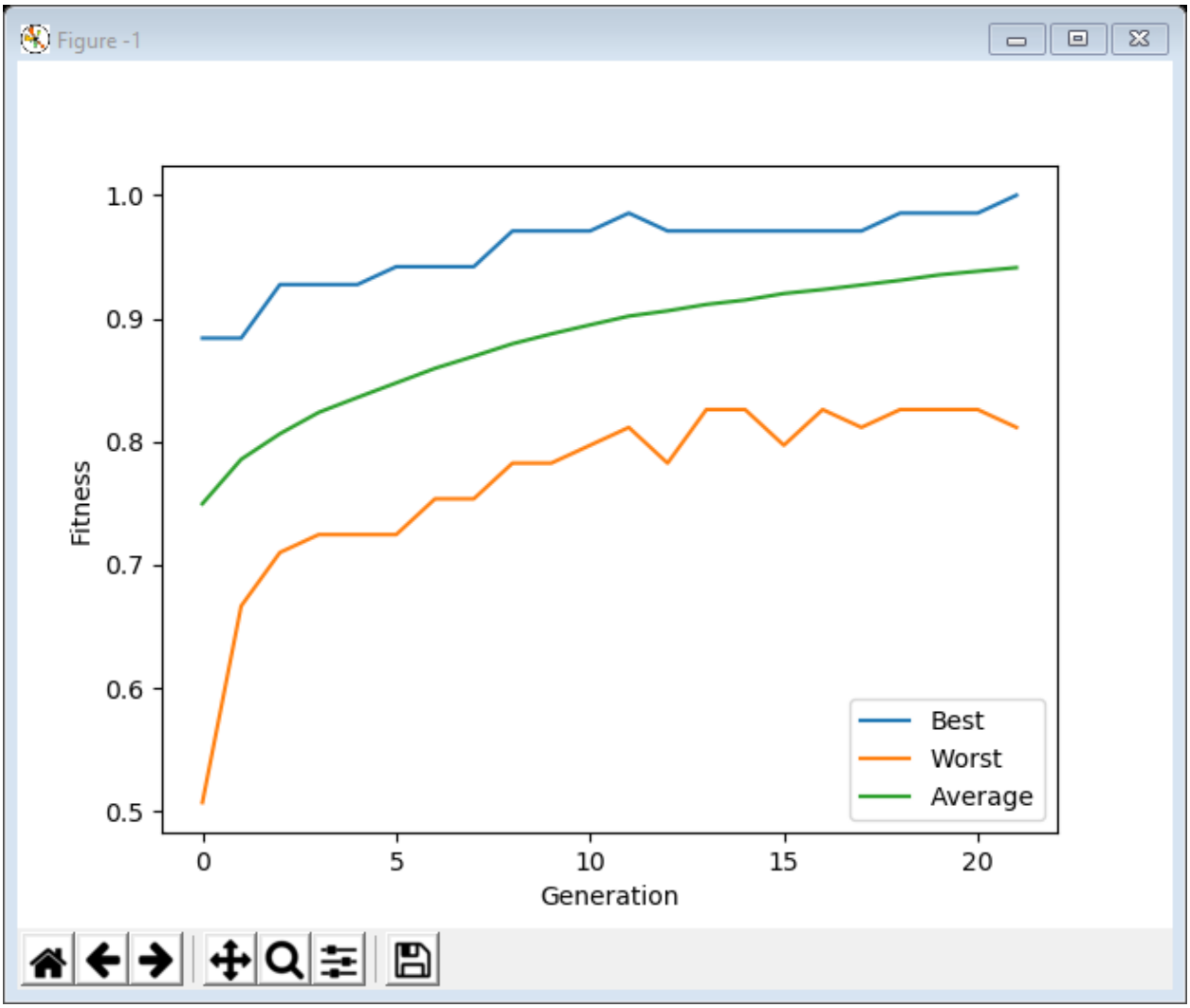
نقشه در آخر به شکل:



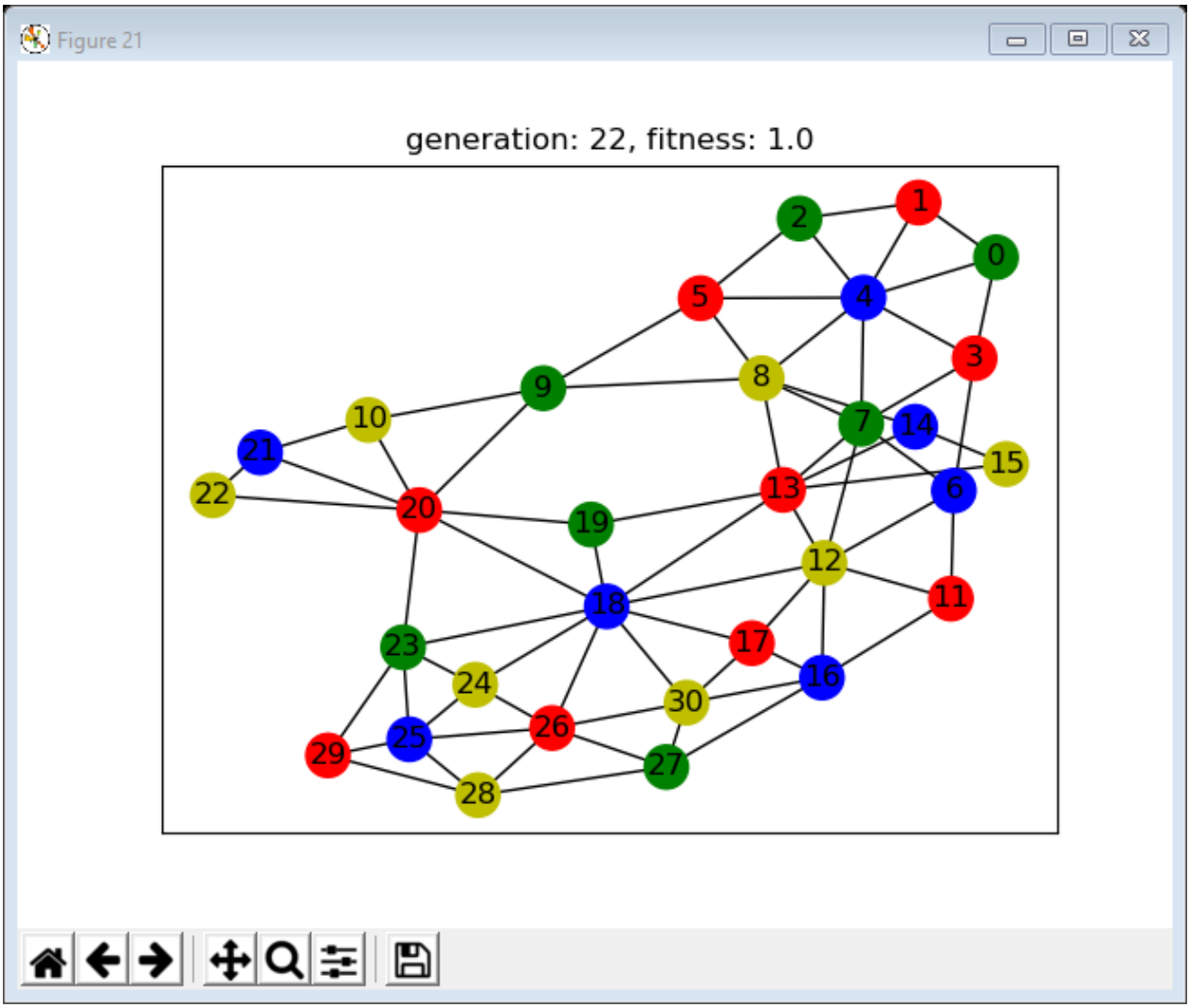
در حالت سوم:

```
population_size = 1000 # number of initial population
muniationRate = 0.01
n_generations = 50 #  times
tornumentSize = 2  # In each generation update a certain number of ran
dom parents is kept
```

در ۲۲ امین نسل به حالت ایده آل میرسیم.



نقشه به شکل:

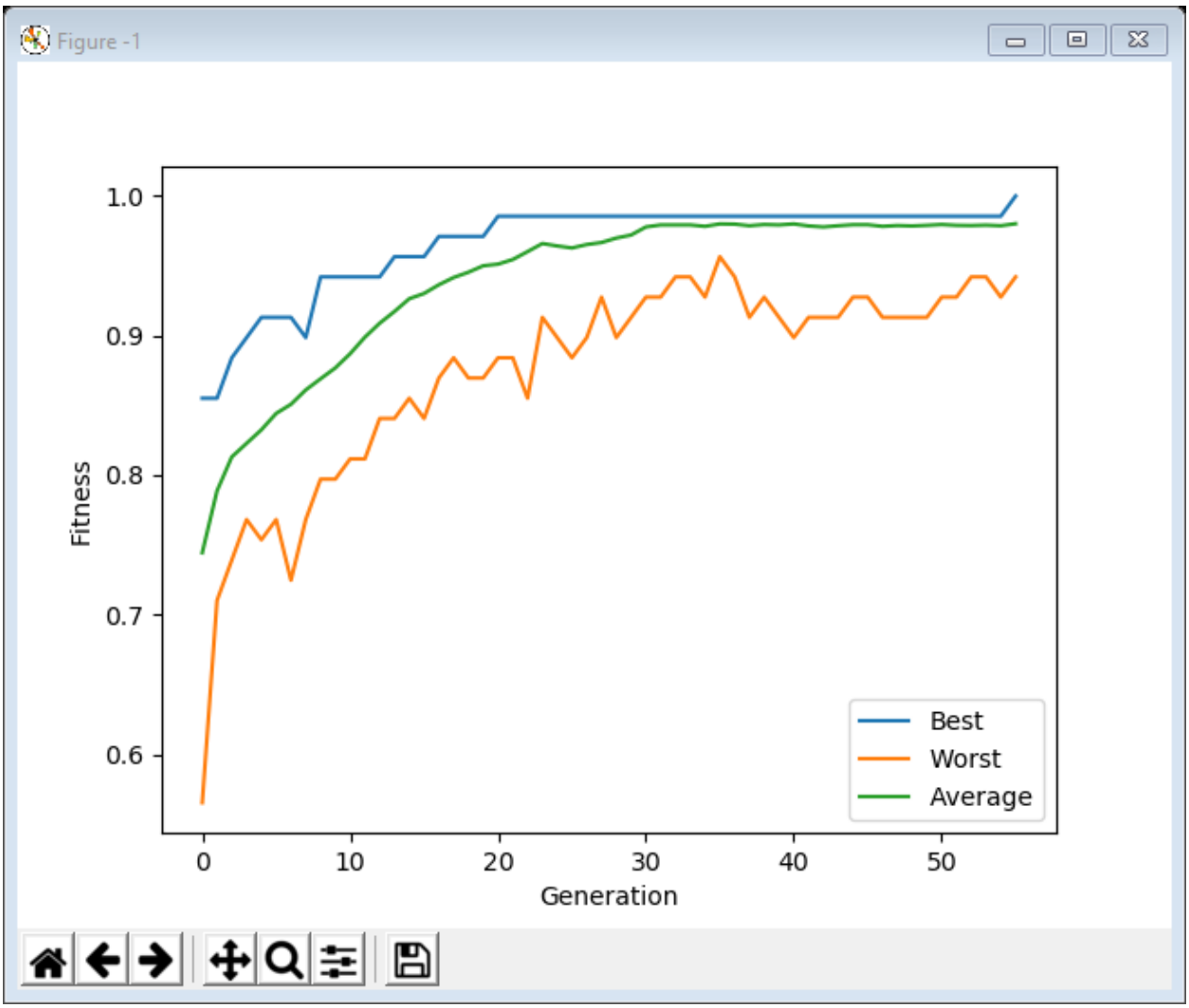


از این سه حالت میتوان فهمید که اگر سایز جمعیت را زیاد (۱۰۰۰) در نظر بگیریم، میتوانیم پس از تعداد کمی نسل(در اینجا ۲۲ نسل) به جواب نهایی برسیم یعنی تابع شایستگی برابر با 1 شو د. البته در بعضی مسائل این زیاد شدن به ما کمک نمیکند و باعث گیج شدن الگوریتم میشود (برای مثال مساله هزارتو با افزایش باعث میشود حالت بسیار شود چون در جلوتر تعداد بیشتر میشود.)

در حالت چهارم:

```
population_size = 100 # number of initial population
muniationRate = 0.01
n_generations = 500 #  times
tornumentSize = 2
```

برای این حالت میتوان فهمید که تاثیر تعداد نسل ها تا جایی از مینیمم لازم کمتر نباشد تاثیری ندارد، بدین صورت که اگر ما تعداد نسل را هم زیاد کنیم باز در وسط الگوریتم به حالت نهایی میرسیم.



نقشه در آخر:

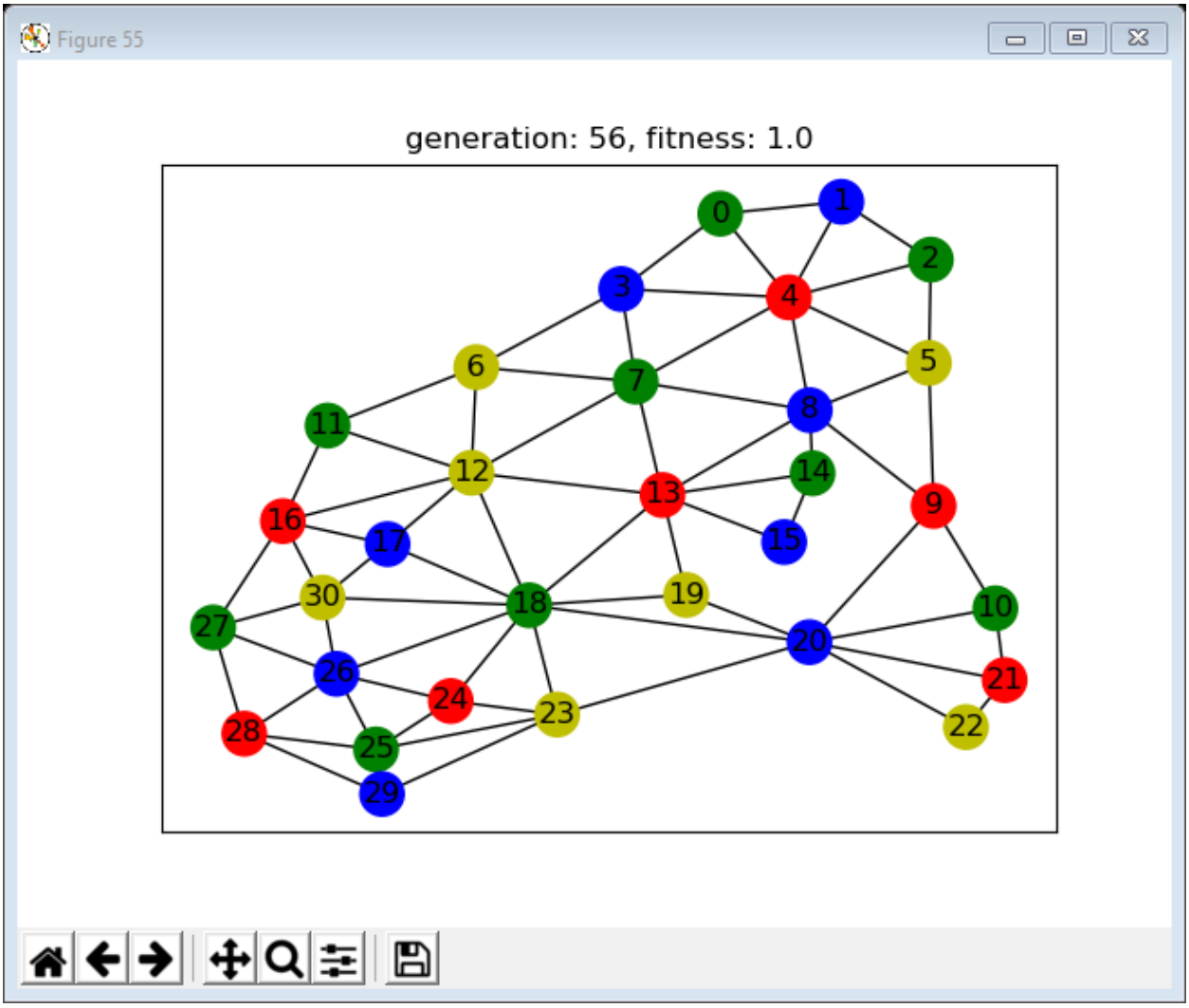
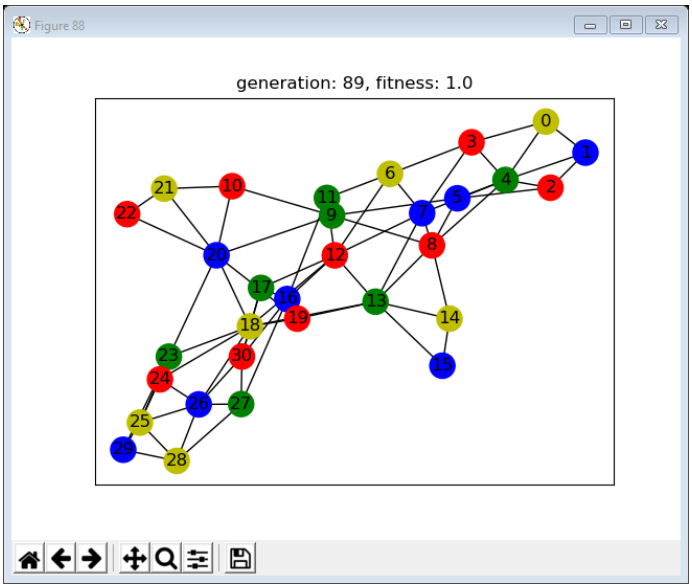


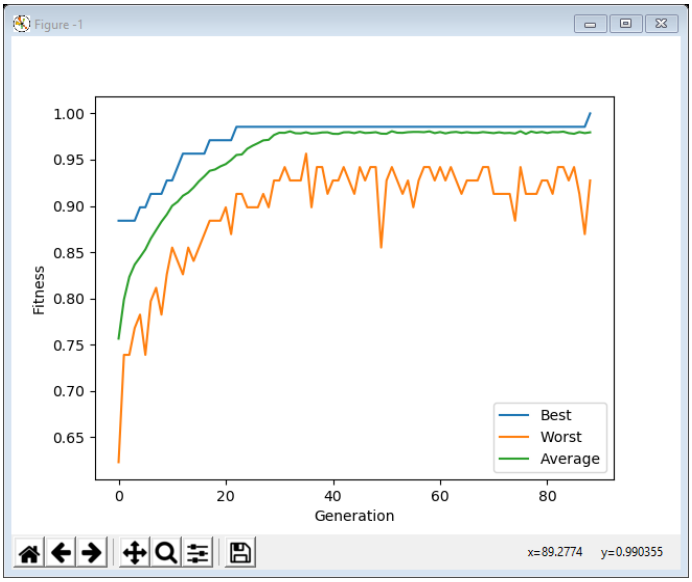
Figure 55
generation: 56, fitness: 1.0
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

در حالت کلی:

جمعیت بزرگتر باعث میشود که بعد از تعداد نسل های کمتری به جواب برسیم چون به طور همزمان مقدار بیشتری از فضای حالت را بررسی می کنیم. البته همانطور که در قبل گفتم در شرایطی تاثیر منفی هم دارد.
اندازه تورنومنت بیشتر باعث تسریع حرکت به سوی جمعیت اصلاح شده مبشود این انتخاب به معنای انتخاب نخبه هاست، اگر تعداد نخبه های انتخاب شده زیاد باشد، باعث میشود تا فرزندهای نامناسب هم انتخاب شود.
در صورت انتخاب کم نخبه (مثلاً ۱) باعث میشود تا اگر این ژن مشکلی داشته باشد برای فرزندان هم تکرار شود و در نسل باقی بماند.
از بین مقادیر داده شده برای Mutation Rate ، مقدار زیاد باعث میشود تا از حالت ایده آل فاصله بگیرد. البته این جهش به فرار از ماکس محلی کمک میکنند.

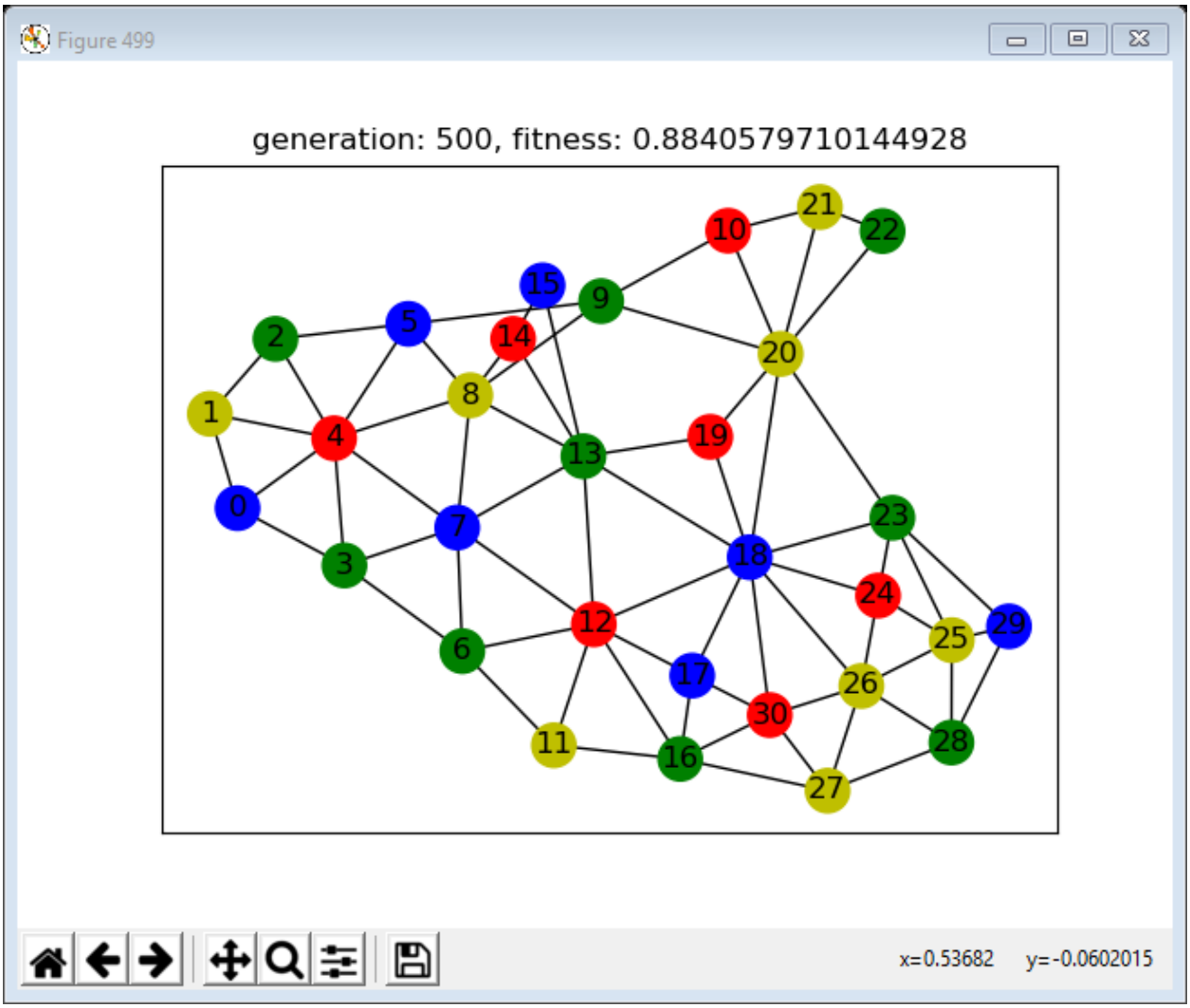
نمودار برای مقدار ۰,۰۱ با ماکس ۱:





برای مقدار ۰,۱ با مقدار فیتنس ۰,۸۸:

همانطورکه میبینیم در نسل ۵۰۰ هم به حالت نهایی نرسیدیم.



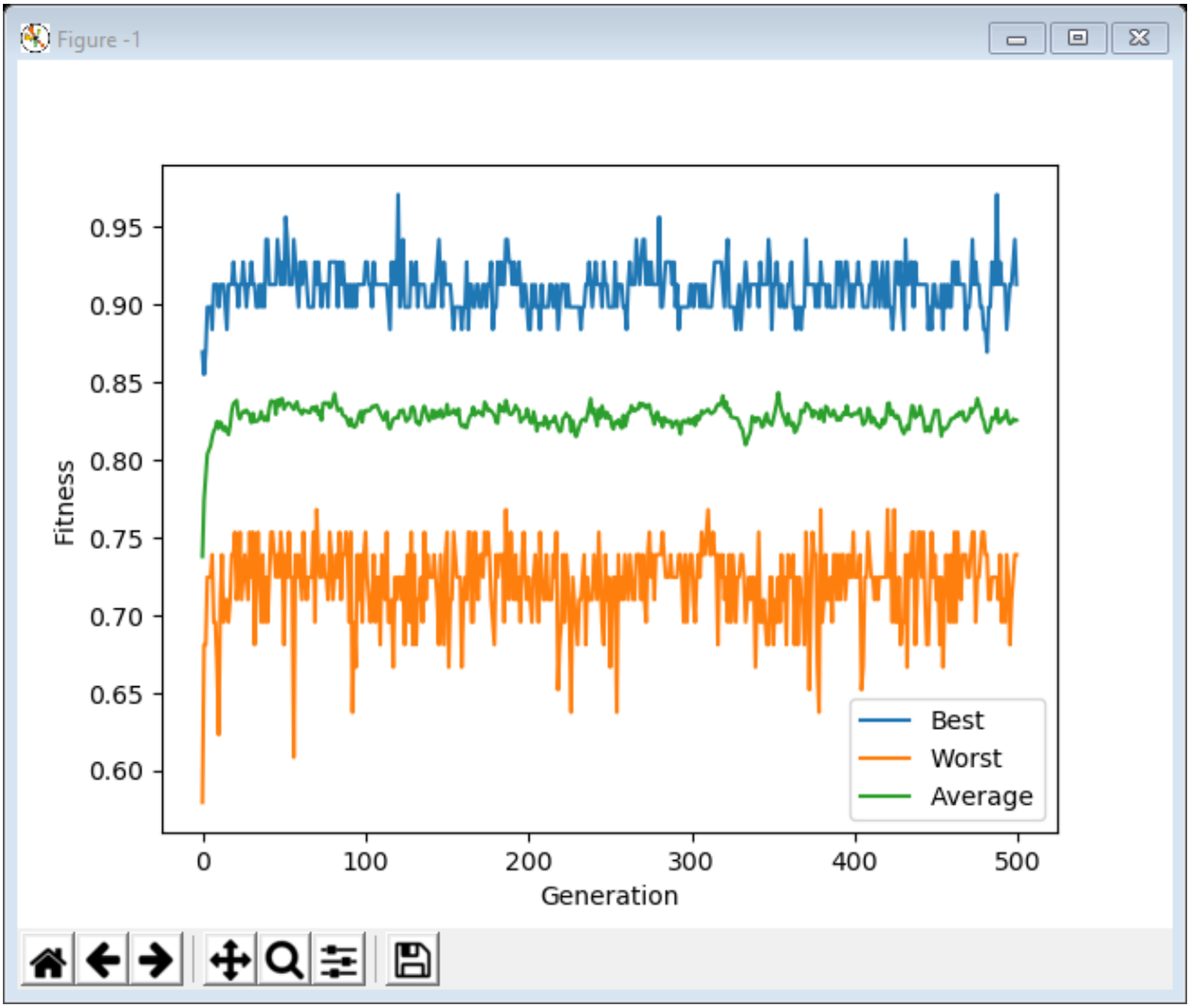


Figure -1
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
Fitness
0
100
200
300
400
500
Generation
Best
Worst
Average

۲. در روش ذوب فلزات اقدام به محاسبه توابع دمای مختلفی کردیم که نتیجه بدین صورت شد:

تابع اول:

$$T_k = T_0\alpha^k, 0.8 \leq \alpha \leq 0.9$$

برای مقدار ۰٫۸ برای α در ثانیه ۳۳۴۰ مقدار دما برابر صفر شد.

تابع دوم:

$$T_k = \frac{T_0}{1 + \alpha \log 1 + k}, \alpha > 1$$

این الگوریتم بسیار کندتر از تابع اول است.

تابع سوم:

$$T_k = \frac{T_0}{1 + \alpha k}, \alpha > 0$$

این تابع از تابع قبلی بهتر عمل میکند.

تابع چهارم:

$$T_k = \frac{T_0}{1 + \alpha k^2}, \alpha > 0$$

این تابع در مقادیر از تابع قبلی نیز بهتر عمل میکند و میشود مقدار α را کوچک گذاشت.