

در ابتدا لازم به ذکر است میتوانید مخزن گیت هاب این پروژه را بر روی لینک زیر ببینید :

https://github.com/mmaghajani/AI_2

مساله هشت وزیر

برای این مساله تابع هدفی به صورت زیر تعریف شده است:

تعداد جفت وزیر هایی که همدیگر را تهدید میکنند باید توجه داشت که این میزان هنگامیکه به یک حالت پایانی برسیم برابر صفر می شود . در نتیجه تابع هدف ما مقدار بالا را در یک منفی ضرب میکند چرا که ارزش حالت هایی که تعداد بیشتری وزیر در آن ها تهدید می شود برای ما کمتر است و این تابع هدف بیانگر ارزش راه حل نیز میباشد.

در این مساله حداکثر تعداد گام ها برابر با ۱۰۰۰۰۰۰۰ گام در نظر گرفته شده است

روش ارایه حالت در این مساله به صورت یک آرایه هشت تایی میباشد که در هر درایه آن نماینده یک ستون در صفحه شطرنج میباشد و شماره سطری که وزیر در آن است در درایه مورد نظر قرار میگیرد. همچنین حالت اولیه برابر حالت زیر است:

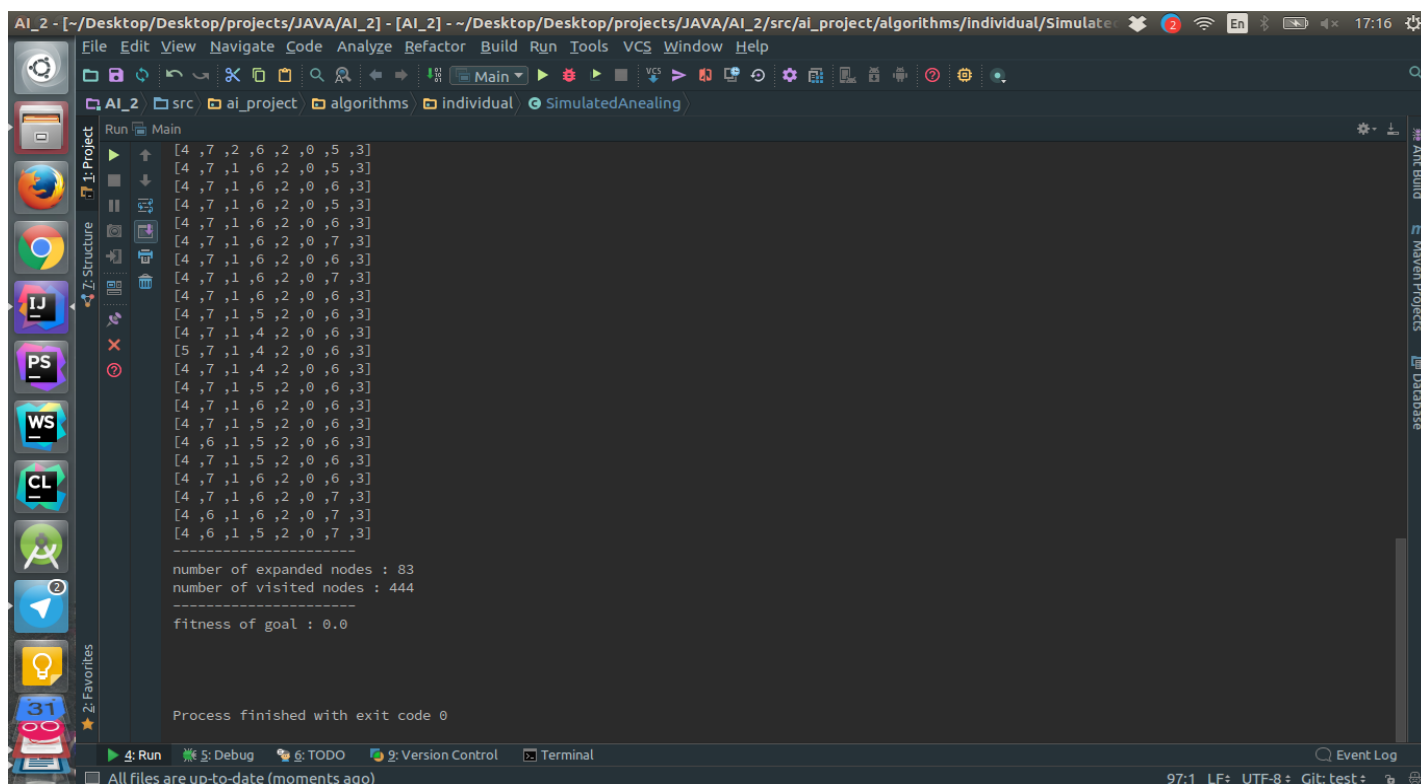
[۰, ۱, ۲, ۳, ۴, ۵, ۶, ۷]

(الف)

برای این قسمت سه روش کاهش در نظر گرفتیم:

متغیر step بیانگر زمان (تعداد گام مساله) میباشد بنابراین روش های زیر در نظر گرفته شده است:

$$t = \frac{|\sin step|}{step} \quad (۱)$$



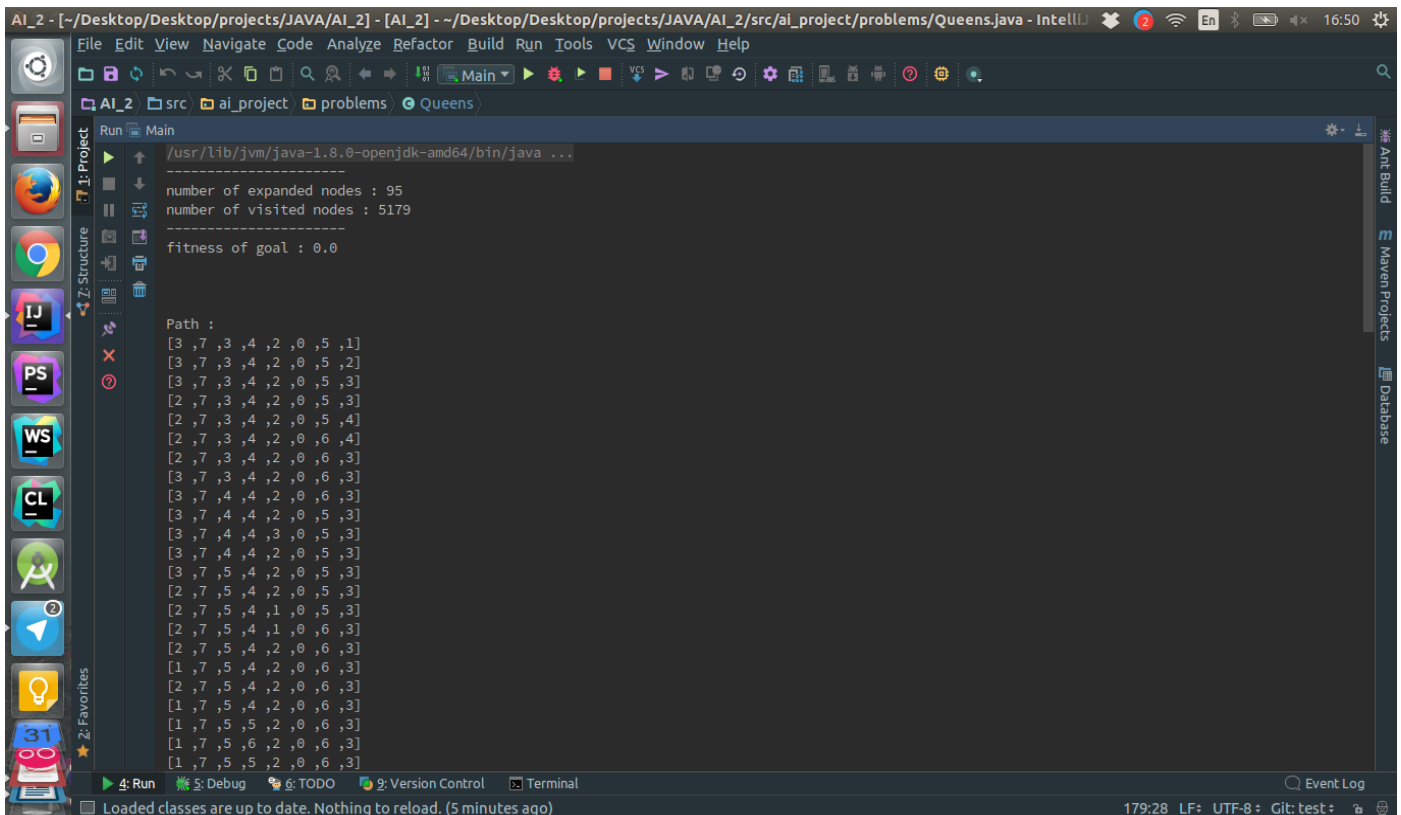
```
AI_2 - [~/Desktop/Desktop/projects/JAVA/AI_2] - [AI_2] - ~/Desktop/Desktop/projects/JAVA/AI_2/src/ai_project/algorithms/individual/Simulate...
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
AI_2 src ai_project algorithms individual SimulatedAnnealing
Run Main
[4, 7, 2, 6, 2, 0, 5, 3]
[4, 7, 1, 6, 2, 0, 5, 3]
[4, 7, 1, 6, 2, 0, 6, 3]
[4, 7, 1, 6, 2, 0, 5, 3]
[4, 7, 1, 6, 2, 0, 6, 3]
[4, 7, 1, 6, 2, 0, 7, 3]
[4, 7, 1, 6, 2, 0, 6, 3]
[4, 7, 1, 6, 2, 0, 7, 3]
[4, 7, 1, 6, 2, 0, 6, 3]
[4, 7, 1, 5, 2, 0, 6, 3]
[4, 7, 1, 4, 2, 0, 6, 3]
[5, 7, 1, 4, 2, 0, 6, 3]
[4, 7, 1, 4, 2, 0, 6, 3]
[4, 7, 1, 5, 2, 0, 6, 3]
[4, 7, 1, 6, 2, 0, 6, 3]
[4, 7, 1, 5, 2, 0, 6, 3]
[4, 6, 1, 5, 2, 0, 6, 3]
[4, 7, 1, 5, 2, 0, 6, 3]
[4, 7, 1, 6, 2, 0, 6, 3]
[4, 7, 1, 6, 2, 0, 7, 3]
[4, 6, 1, 6, 2, 0, 7, 3]
[4, 6, 1, 5, 2, 0, 7, 3]
-----
number of expanded nodes : 83
number of visited nodes : 444
-----
fitness of goal : 0.0
Process finished with exit code 0
```

خروجی نمونه در بالا آورده شده است.

در این روش میانگین تعداد راس های expand شده حدود ۱۰۰۰ راس بود. همچنین همواره به جواب نهایی دست پیدا میکردیم

$$t = \left(\frac{1}{step}\right) \quad (۲)$$

در این حالت خروجی نمونه به صورت زیر است:



```
Run Main
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
-----
number of expanded nodes : 95
number of visited nodes : 5179
-----
fitness of goal : 0.0

Path :
[3, 7, 3, 4, 2, 0, 5, 1]
[3, 7, 3, 4, 2, 0, 5, 2]
[3, 7, 3, 4, 2, 0, 5, 3]
[2, 7, 3, 4, 2, 0, 5, 3]
[2, 7, 3, 4, 2, 0, 5, 4]
[2, 7, 3, 4, 2, 0, 6, 4]
[2, 7, 3, 4, 2, 0, 6, 3]
[3, 7, 3, 4, 2, 0, 6, 3]
[3, 7, 4, 4, 2, 0, 6, 3]
[3, 7, 4, 4, 2, 0, 5, 3]
[3, 7, 4, 4, 3, 0, 5, 3]
[3, 7, 4, 4, 2, 0, 5, 3]
[3, 7, 5, 4, 2, 0, 5, 3]
[2, 7, 5, 4, 2, 0, 5, 3]
[2, 7, 5, 4, 1, 0, 5, 3]
[2, 7, 5, 4, 1, 0, 6, 3]
[2, 7, 5, 4, 2, 0, 6, 3]
[1, 7, 5, 4, 2, 0, 6, 3]
[2, 7, 5, 4, 2, 0, 6, 3]
[1, 7, 5, 4, 2, 0, 6, 3]
[1, 7, 5, 5, 2, 0, 6, 3]
[1, 7, 5, 6, 2, 0, 6, 3]
[1, 7, 5, 5, 2, 0, 6, 3]
```

با این روش به حالت پایانی زیر (در یکی از اجراها) رسیدیم:

$$[۲, ۶, ۱, ۷, ۴, ۰, ۳, ۵]$$

در این روش میانگین تعداد راس های expand شده حدود ۳۰۰۰ راس است و تعداد راس های مشاهده شده حدود ۲۰ الی ۳۰ هزار است.

$$t = -step + ۱۰۰۰ \quad (۳)$$

خروجی نمونه:

در این روش با چندین بار اجرا هیچ گاه به جواب نرسیدیم و ارزشمندترین راه حلی که بدست آوردیم ارزش آن ۱- بود و این کار با expand کردن حدود یک میلیون راس بدست میآمد که میزان بالایی ست. میانگین ارزش راس حل ها حدود ۲- بود.

نتیجه : روش اول در مجموع بهترین روش میباشد زیرا تعداد راس های کمی را گسترش میدهد و همچنین همواره به جواب میرسد بعد از آن روش دوم بهتر است و سپس روش سوم با توجه به خطی بودن آن اصلاً مطلوب نیست.

(ب)

حال به سراغ اجرای دیگر الگوریتم ها میرویم :

(۱) تپه نوردی معمولی :

در این حالت خروجی به صورت زیر است :

```
AI_2 - [~/Desktop/Desktop/projects/JAVA/AI_2] - [AI_2] - ~/Desktop/Desktop/projects/JAVA/AI_2/src/ai_project/Main.java - IntelliJ IDEA 2016.2
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
AI_2 src ai_project Main
Run Main
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
Path :
[0, 1, 2, 3, 4, 5, 6, 7]
[1, 1, 2, 3, 4, 5, 6, 7]
[1, 0, 2, 3, 4, 5, 6, 7]
[1, 0, 1, 3, 4, 5, 6, 7]
[2, 0, 1, 3, 4, 5, 6, 7]
[2, 0, 1, 4, 4, 5, 6, 7]
[2, 0, 1, 4, 5, 5, 6, 7]
-----
number of expanded nodes : 6
number of visited nodes : 99
-----
fitness of goal : -6.0

Process finished with exit code 0
20:1 LF: UTF-8 Git: test
```

همانطور که پیداست فوراً در یک بهینه محلی گیر میکند و جواب باارزشی هم بدست نمی دهد.

(۲) تپه نوردی تصادفی

در این حالت یکی از خروجی ها به صورت زیر است:

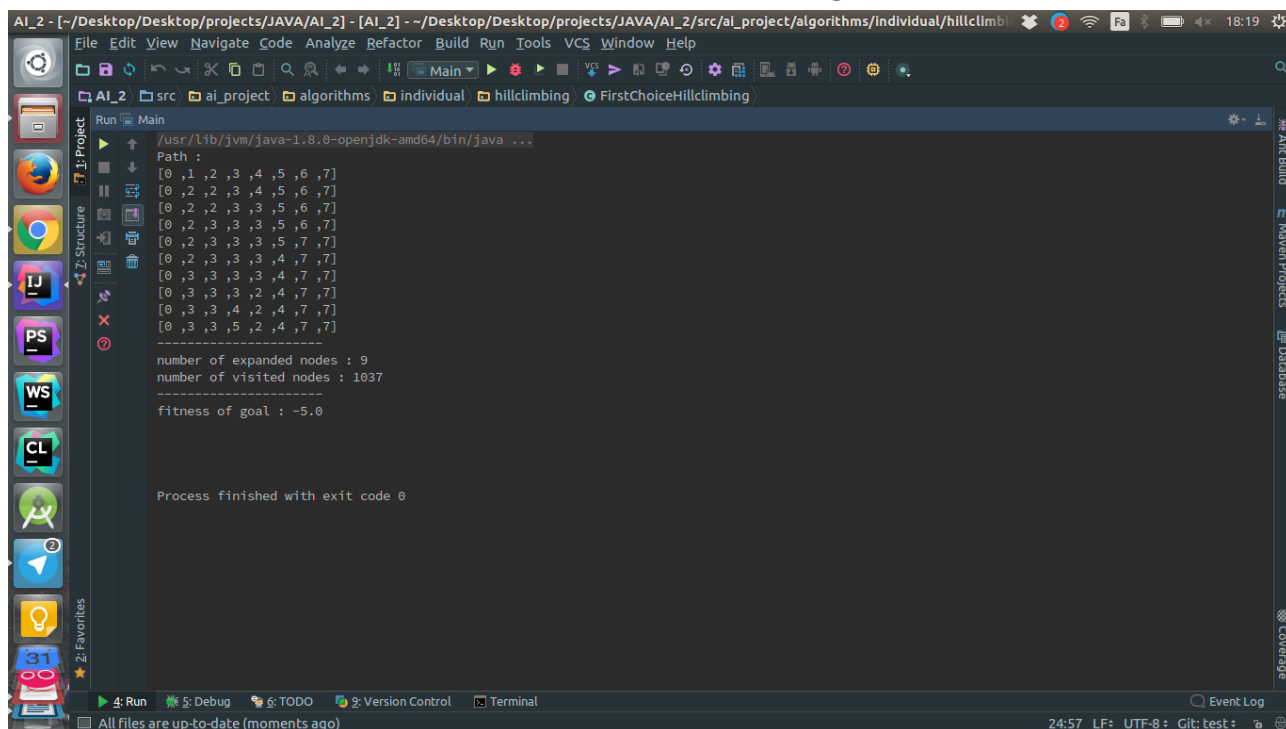
```
AI_2 - [~/Desktop/Desktop/projects/JAVA/AI_2] - [AI_2] - ~/Desktop/Desktop/projects/JAVA/AI_2/src/ai_project/Main.java - IntelliJ IDEA 2016.2
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
AI_2 src ai_project Main
Run Main
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
Path :
[0, 1, 2, 3, 4, 5, 6, 7]
[0, 1, 2, 3, 3, 5, 6, 7]
[0, 1, 2, 3, 3, 5, 6, 6]
[0, 2, 2, 3, 3, 5, 6, 6]
[0, 2, 2, 3, 3, 6, 6, 6]
[0, 2, 2, 3, 3, 6, 7, 6]
[0, 3, 2, 3, 3, 6, 7, 6]
[0, 3, 2, 3, 3, 6, 7, 5]
[0, 4, 2, 3, 3, 6, 7, 5]
[0, 4, 1, 3, 3, 6, 7, 5]
-----
number of expanded nodes : 9
number of visited nodes : 143
-----
fitness of goal : -4.0

Process finished with exit code 0
23:1 LF: UTF-8 Git: test
```

همانطور که معلوم است گاهی اوقات نتایج ضعیف تر از حالت معمولی میدهد اما عموماً به صورت میانگین ارزش جواب حدود ۴- میباشد و بهترین نتیجه حاصله ۳- بود. تعداد راس های گسترش داده شده نیز حدود ۹ الی ۱۰ میباشد.

(۳) تپه نوردی اولین انتخاب

در این حالت تعداد حالات بعدی که به صورت تصادفی تولید می شود توسط ثابت RANDOM_RATE مشخص میگردد که برای این الگوریتم برابر ۱۰۰۰ در نظر گرفته شده است و اگر تعداد همسایه های تولید شده از این تعداد بیشتر شد الگوریتم این حالت را به عنوان گیر افتادن در یک بهینه محلی تلقی میکند. میانگین تعداد گره های گسترش داده شده در این حالت حدود ۱۰ گره است و یک خروجی نمونه به صورت زیر است:



```
Path :
[0, 1, 2, 3, 4, 5, 6, 7]
[0, 2, 2, 3, 4, 5, 6, 7]
[0, 2, 2, 3, 3, 5, 6, 7]
[0, 2, 3, 3, 3, 5, 6, 7]
[0, 2, 3, 3, 3, 5, 7, 7]
[0, 2, 3, 3, 3, 4, 7, 7]
[0, 3, 3, 3, 3, 4, 7, 7]
[0, 3, 3, 3, 2, 4, 7, 7]
[0, 3, 3, 4, 2, 4, 7, 7]
[0, 3, 3, 5, 2, 4, 7, 7]

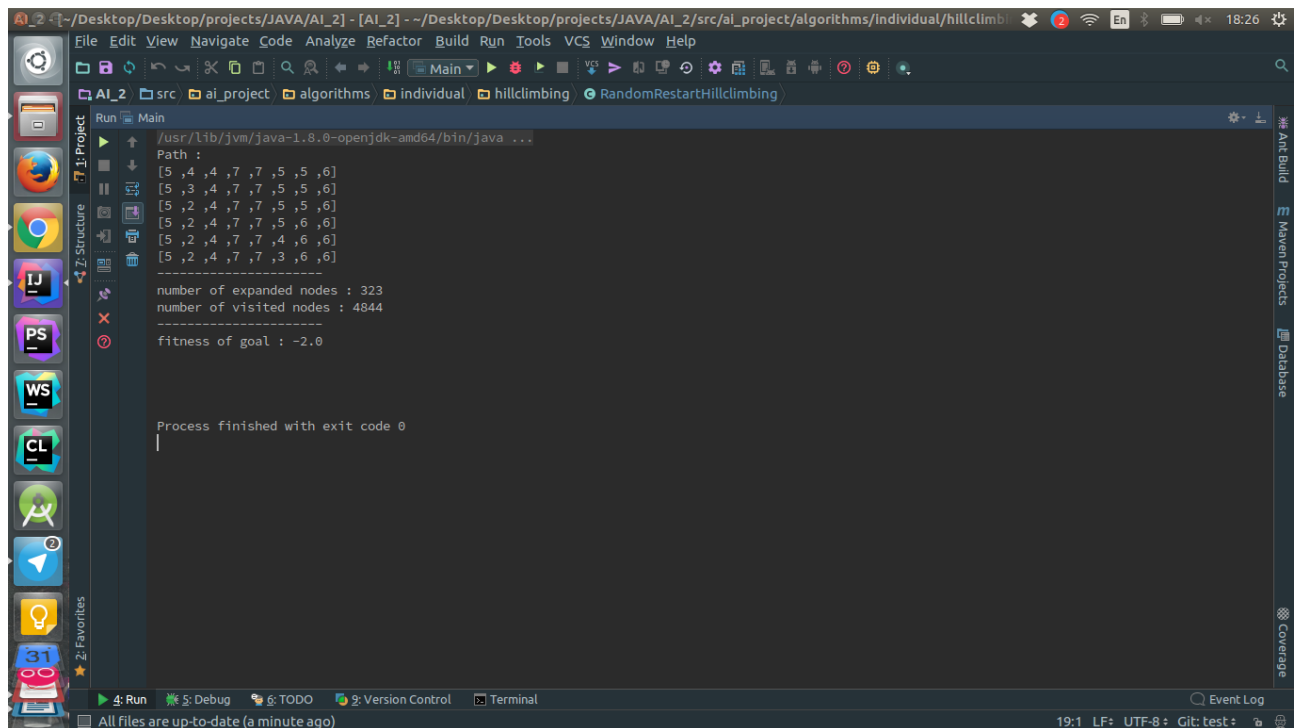
-----
number of expanded nodes : 9
number of visited nodes : 1037
-----
fitness of goal : -5.0

Process finished with exit code 0
```

میانگین ارزش راه حل های تولید شده ۵- بود و بهترین جواب تولید شده ارزشی برابر با ۴- داشت که درواقع از الگوریتم قبلی بدتر میباشد.

(۴) تپه نوردی با شروع مجدد تصادفی

در این حالت تعداد شروع مجدد ها توسط ثابت RESTART_LIMIT کنترل می شود که برابر با ۱۰۰ است. یکی از خروجی های نمونه به صورت زیر است:



```
Run: Main
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
Path :
[5, 4, 4, 7, 7, 5, 5, 6]
[5, 3, 4, 7, 7, 5, 5, 6]
[5, 2, 4, 7, 7, 5, 5, 6]
[5, 2, 4, 7, 7, 5, 6, 6]
[5, 2, 4, 7, 7, 4, 6, 6]
[5, 2, 4, 7, 7, 3, 6, 6]
-----
number of expanded nodes : 323
number of visited nodes : 4844
-----
fitness of goal : -2.0

Process finished with exit code 0
```

در این روش ارزشمند ترین جوابی که تولید شد همان جواب بهینه بود و در بین تمامی الگوریتم های تپه نوردی تنها نسخه ای که موفق به تولید جواب بهینه شد همین الگوریتم بود. میانگین ارزش جواب ها نیز بین ۲- تا ۳- بود که از این حیث نیز بهترین الگوریتم تپه نوردی به شمار میرفت و همچنین میانگین گره های گسترش داده شده حدود ۳۰۰ گره بود که تعداد بیشتری در مقایسه با بقیه نسخه های تپه نوردی میباشد.

نتیجه : همانطور که دیدید در بین الگوریتم های تپه نوردی ، الگوریتم با شروع مجدد تصادفی در مجموع بهترین نتایج را بدست میداد و کمتر در بهینه های محلی گیر می افتاد اما الگوریتم سرد کردن تدریجی با تابع کاهش $t = |\sin(\text{step})| / \text{step}$ حتی از این الگوریتم بهتر عمل میکرد زیرا همواره جواب بهینه را بدست میداد اما از لحاظ تعداد گره های گسترش داده حدوداً سه برابر الگوریتم تپه نوردی با شروع مجدد تصادفی گره گسترش میداد که اندکی آن را کند کرده بود.

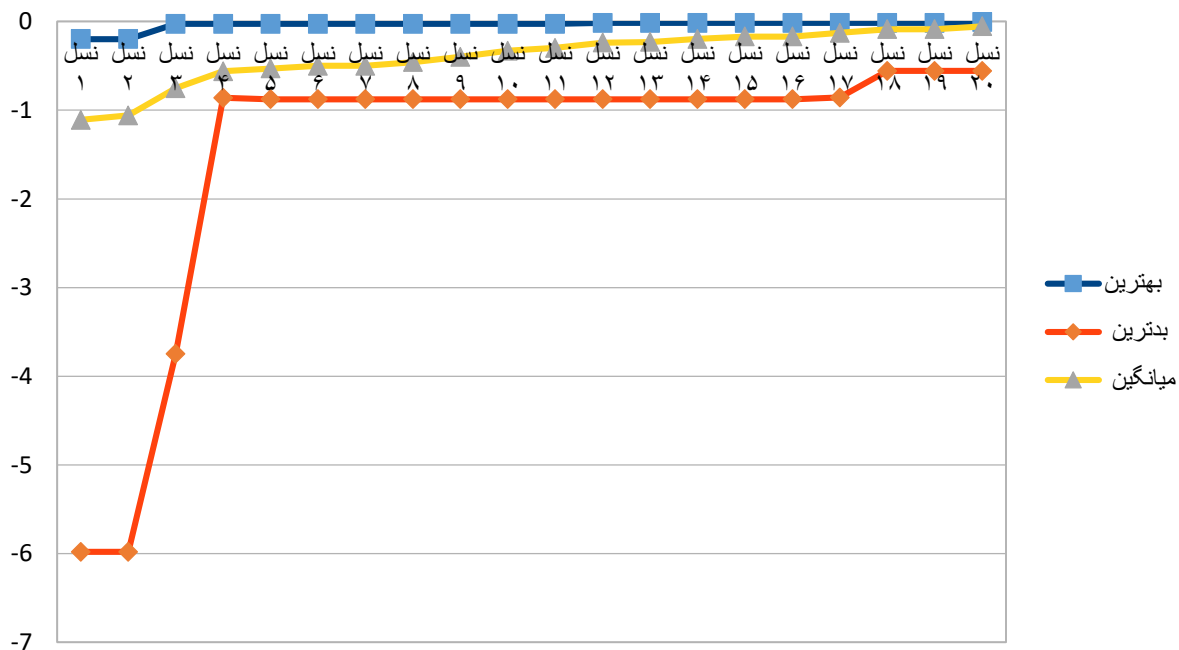
مساله حل معادله :

در این مساله ابتدا معادله را به حالت استاندارد در آورديم يعنى همه جملات را به يك طرف تساوى برديم در نتيجه براى پيدا كردن جواب بايد به دنبال صفر كردن تابع بدست آمده باشيم. حال براى بدست آوردن تابع هدف اينگونه عمل ميكنيم كه مقدار اين تابع را ابتدا قدر مطلق ميگيرم و سپس در يك منفي ضرب ميكنيم با اين كار در واقع ميزان اختلاف ما با جواب بدست ميآيد و چون هر چه اختلاف كمتر باشد ارزش جواب بيشتر است پس بايد آن را در يك منفي ضرب كردن تا تابع هدف با تابع شايستگي يكسان گردد.

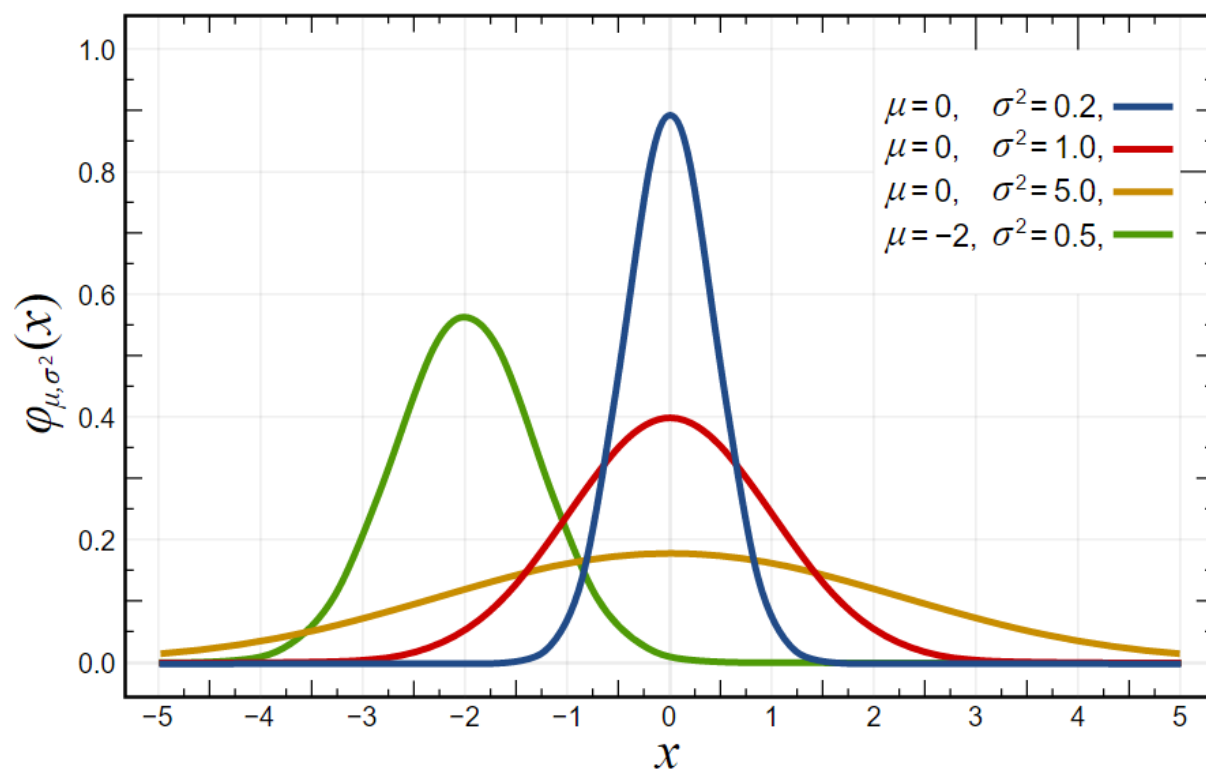
در اين مساله براى توليد حالات اوليه از تابع رندم به صورتى استفاده شده است كه اعدادى در بازه گفته شده توليد كند. همچنين با توجه به پيوسته بودن فضاى مساله براى تابع next state كه حالات بعدى را توليد ميكند در جهت گرايدان تابع داده شده حركت مى شود. براى تشخيص حالت نهايى نيز يك دقت جواب در نظر گرفته شده كه توسط ثابت PRECISION_SOLVING_EQUATION كنترل ميشود (اين مقدار به صورت پيشفرض برابر ۰.۰۱ است).

الف)

نمودار بهترين و بدترين و ميانگين شايستگي :



ب) با توجه به نمودار زير همانطور كه مي بيند اگر واريانس از يك حدى بيشتر شود ديگر جواب هاى جهش داده شده در بازه قرار نميگيرد در نتيجه همگرابى نسل به سمت جواب هاى بهينه كند كتر ميشود.



ج) در این بخش تعداد کل ارزیابی ها را ۱۰ گذاشتیم

برای بخش بهترین های هر نسل با افزایش تعداد جمعیت بهترین ها نیز بهتر میشدند و شایستگی بیشتری داشتند مثلاً وقتی جمعیت را برابر ۱۰۰ در نظر گرفتیم هیچ گاه در پایان ارزیابی های شایستگی (۱۰ مرحله) ارزش بهترین نسل از ۰,۰۵- بیشتر نبود . اما از طرفی با افزایش جمعیت بدترین های هر نسل نیز بدتر می شدند و با افزایش جمعیت میانگین شایستگی در نسل ها نیز اندکی رو به بدتر شدن میرفت که البته خیلی قابل توجه نبود.