داکیومنت کد الگوریتم ژنتیک را برای حل مسأله بهینه‌سازی مسیر یک جاده در یک نقشه است. در ادامه، توضیحاتی در مورد توابع و اپراتورهای موجود در کد ارائه خواهم داد:

پارامترها :

- population_size: تعداد افراد موجود در هر نسل از الگوریتم ژنتیک.
- mutation_rate: احتمال جهش یک ژن در هر نسل.
- crossover_rate: احتمال تلاقی دو افراد در هر نسل.
- max_generations: تعداد حداکثر نسل‌هایی که الگوریتم ژنتیک اجرا می‌شود.

پارامتر های نقشه جاده :

- map_size: ابعاد نقشه به صورت تاپل (سطح افقی، سطح عمودی).
- roughness_levels: سطوح مختلف زمین‌ناپیچیده برای انتخاب به عنوان ویژگی هر نقطه از نقشه.
- roughness_probabilities: احتمالات مربوط به هر سطح زمین که برای ایجاد نقشه جاده تصادفی استفاده می‌شوند.

توابع :

تابع create_road_mapدر این کد مسئول ایجاد نقشه جاده تصادفی است. این تابع به صورت زیر تعریف شده است:

در این تابع , از تابع np.random.choice در کتابخانه NumPy استفاده می‌شود. این تابع به شما اجازه می‌دهد بر اساس توزیع احتمالاتی داده شده، انتخاب‌های تصادفی را ایجاد کنید. در اینجا، roughness_levels نشان دهنده سطوح مختلف جاده است و roughness_probabilitiesمقادیر احتمالی مربوط به هر سطح را مشخص می‌کند. با استفاده از np.random.choice، یک آرایه با ابعاد map_sizeایجاد می‌شود که هر عنصر آن یک سطح خشنی تصادفی از بین سطوح مختلف است.
سپس نقشه جاده تصادفی ساخته شده در road_mapذخیره می‌شود و از تابع returnبرگشت داده می‌شود.
به طور کلی، این تابع برای ایجاد یک نقشه جاده تصادفی با استفاده از سطوح خشنی تعیین شده و ابعاد مشخص شده استفاده می‌شود.

تابع calculate_costدر این کد برای محاسبه هزینه ایجاد جاده بین دو مختصات (نقطه) استفاده می‌شود. این تابع به صورت زیر تعریف شده است:

در این تابع، دو مختصات coord1و coord2 به عنوان ورودی دریافت می‌شوند. این مختصات نشان دهنده دو نقطه در نقشه جاده است.
ابتدا، با استفاده از شرط 1=> (abs(coord1[1] - coord2[1]) <= 1 and abs(coord1[0] - coord2[0])بررسی می‌شود که آیا دو نقطه در مجاورت یکدیگر هستند یا خیر. اگر این شرط برقرار باشد، هزینه جاده برابر با مقدار road_map[coord2[0], coord2[1]] خواهد بود. در واقع، هزینه جاده برابر با سطح نقطه coord2در نقشه جاده است.در این صورت، فاصله بین coord1و coord2محاسبه می‌شود. فاصله در اینجا به عنوان مجموع مربعات اختلاف مختصات در جهت افقی و عمودی محاسبه می‌شود. به علاوه، هزینه جاده نیز با اضافه کردن سطح نقطه coord2در نقشه جاده محاسبه می‌شود. در نهایت، مقدار هزینه جاده به عنوان خروجی تابع returnداده می‌شود.
به طور خلاصه، تابع calculate_costبرای محاسبه هزینه ایجاد جاده بین دو نقطه بر اساس فاصله و سطح نقطه‌ی مقصد استفاده می‌شود.

تابع **calculate_fitness** در الگوریتم ژنتیک شما مسئول محاسبه ارزش فیتنس (fitness) یک فرد (individual) در جمعیت است. در اینجا، فیتنس به عنوان یک معیار برای ارزیابی ژنوم یا مسیر ایجاد شده توسط الگوریتم ژنتیک استفاده می‌شود. بر اساس ارزش فیتنس، فردی که مسیر بهینه‌تری ارائه داده باشد، ارزش فیتنس بالاتری دارد.

در تابع **calculate_fitness**، شما از مجموع هزینه (cost) مسیر برای محاسبه ارزش فیتنس استفاده می‌کنید. مسیر بین هر دو نقطه از جمله‌های مسیر با استفاده از **calculate_cost** محاسبه شده و مجموع هزینه‌ها به عنوان ارزش فیتنس برگردانده می‌شود.

در واقع، این تابع ارزش مسیرهای مختلف را بر اساس هزینه‌های مختلف محاسبه می‌کند و این ارزشها را برای انتخاب والدین و ارزیابی جمعیت استفاده می‌کنید.

تابع **generate_population**در الگوریتم ژنتیک شما مسئول تولید جمعیت اولیه از افراد (جاده‌ها) است. این تابع افراد را به صورت تصادفی در فضای جستجو (فضای مسئله) ایجاد می‌کند. در ادامه نحوه عملکرد این تابع توضیح داده شده است:
تابع یک لیست به نام **population**ایجاد می‌کند. در هر مرحله یک فرد (جاده) جدید تولید می‌شود و به لیست اضافه می‌شود. این فرد با استفاده از تابع **random.randint(0, map_size[0]-1)**برای تولید مختصات y و از **i** برای تولید مختصات محورx به صورت تصادفی ایجاد می‌شود. این مراحل برای تمام افراد جمعیت تکرار می‌شود و لیست جمعیت ایجاد شده به عنوان خروجی تابع برگردانده می‌شود.

به این ترتیب، تابع **generate_population**وظیفه تولید یک جمعیت اولیه از افراد (جاده‌ها) را با موقعیت‌های تصادفی در نقشه انجام می‌دهد.

تابع **mutate**در الگوریتم ژنتیک شما، برای اعمال جهش به افراد جمعیت (جاده‌ها) استفاده می‌شود. جهش یک عملیات تصادفی است که با احتمال مشخص (mutation_rate) اعمال می‌شود و باعث تغییر تصادفی در ایندیوید (فرد) می‌شود. در ادامه نحوه عملکرد این تابع توضیح داده شده است:

این تابع دارای ورودی است یعنی یک جاده را به عنوان ورودی میپذیرد و در مرحله بعد تابع برای هر نقطه در مسیر (هر گره از جاده)، با احتمال **mutation_rate**یک جهش اعمال می‌کند. اگر جهش اعمال شود، مختصات y تصادفی جدید انتخاب می‌شود و مختصات x حفظ می‌شود. این جهش باعث تغییر تصادفی یک نقطه در جاده می‌شود. و به این ترتیب، تابع **mutate**وظیفه تغییر تصادفی یک نقطه در جاده با احتمال مشخص را انجام می‌دهد.

تابع **mutate1**نیز مانند **mutate**برای اعمال جهش به افراد جمعیت (جاده‌ها) در الگوریتم ژنتیک استفاده می‌شود. این تابع از تابع **mutate**با یک تفاوت اساسی متمایز می‌شود. در اینجا توضیح نحوه عملکرد این تابع آورده شده است:

این تابع دارای ورودی است یعنی یک جاده را به عنوان ورودی میپذیرد و تفاوت اصلی این تابع با تابع **mutate**در اینجاست که فقط یک نقطه از جاده را با اندیس i انتخاب می‌کند و تغییرات را فقط بر روی این نقطه انجام می‌دهد. این نقطه با احتمال **mutation_rate**انتخاب شده و مختصات y تصادفی جدید انتخاب می‌شود و مختصات x حفظ می‌شود. بنابراین، تفاوت اصلی میان **mutate**و **mutate1**در تعداد نقاطی است که تحت جهش قرار می‌گیرند. در **mutate**، همه نقاط ممکن است تحت جهش قرار گیرند، در حالی که در **mutate1**تنها یک نقطه انتخاب می‌شود.

تابع **crossover**در الگوریتم ژنتیک برای انجام عملیات تلاقی بین دو والد (جاده) به منظور ایجاد فرزند استفاده می‌شود. در این تابع، هر نقطه‌ای که تحت تلاقی قرار بگیرد، با احتمال **crossover_rate**انتخاب شده از والد متفاوت انتخاب می‌شود. تا این نقطه، والد اول به عنوان والد فرزند انتخاب شده و سپس از نقطه انتخاب شده به بعد، مقدار این نقطه تا انتهای جاده با مقدار معادل در همان مکان از والد دوم جایگزین می‌شود.

مراحل اصلی تابع **crossover**به شرح زیر است:

ورودی : تابع دو والد **parent1** و **parent2** را به عنوان ورودی می‌پذیرد.

تلاقی جاده‌ها: برای هر نقطه از جاده‌ها، با احتمال **crossover_rate**چک می‌شود که آیا این نقطه باید از والد دوم درآید یا نه. اگر شرط برقرار باشد، مقدار نقطه معادل در همان مکان از والد دوم (**parent2**) جایگزین می‌شود.

خروجی :جاده فرزند حاصل از تلاقی به عنوان خروجی تابع برگردانده می‌شود.

در این تابع، امکان دارد فرزند به نحوی از والدان بهتر باشد (در صورتی که یک نقطه موثر تلاقی شده باشد) یا ممکن است ویژگی‌های خاص والدان با هم ترکیب شوند. این تلاقی به الگوریتم کمک می‌کند تا از تنوع ژنتیک بهره‌مند شود و به جستجوی موثرتر برای راه حل‌های بهتر برسد.

تابع **crossover1**مشابه تابع **crossover**عمل می‌کند با این تفاوت که به جای تلاقی تمام جاده‌ها از یک نقطه، فقط از یک نقطه‌ی تصادفی شروع می‌شود و تمام اعضای بعدی جاده از والد دوم به عنوان فرزند منتقل می‌شوند.

در تابع **crossover1**، با احتمال **crossover_rate**یک نقطه تصادفی برای شروع تلاقی انتخاب می‌شود .(**i**) سپس تمام اعضای جاده از این نقطه به بعد با مقدار معادل در همان مکان از والد دوم (**parent2**) جایگزین می‌شوند. اگر با **crossover**تمام جاده‌ها تحت تلاقی قرار می‌گرفتند، حالا با **crossover1**فقط بخشی از جاده از والد دوم می‌آید و بقیه از والد اول. این نوع تلاقی می‌تواند به مفهوم‌پذیری جاده‌ها کمک کند و تاثیر مستقیم براحتی محدود شود به بخش خاصی از جاده. این ویژگی معمولاً وابسته به مسئله مورد نظر و نحوه نمایش راه‌حل در ژنتیک است.

تابع **select_parents**به منظور انتخاب والدین برای تولید نسل بعدی در الگوریتم ژنتیک استفاده می‌شود. این تابع با استفاده از روش تورنمنت (tournament selection)، دو والدین انتخاب می‌کند .مراحل اجرای تابع به شرح زیر است :

برای هر والد مجموعه‌ای از فردهای جمع‌آوری می‌شود. تعداد این فردها به عنوان اندازه تورنمنت تعیین می‌شود. در تابع **select_parents** از تعداد 3 تورنمنت استفاده شده است ' سپس بر اساس تابع فیتنس **calculate_fitness** مقدار فیتنس هر فرد در تورنمنت محاسبه می‌شود و فرد برنده، یعنی فرد با بیشترین فیتنس در تورنمنت، به عنوان والد انتخاب می‌شود. این مراحل برای انتخاب دو والد به صورت تکراری اجرا می‌شوند.

در نهایت، تابع **select_parents** لیستی حاوی دو والدین انتخاب شده را بازمی‌گرداند. این والدین سپس برای تولید فرزندان در مرحلهٔ بعد الگوریتم ژنتیک استفاده می‌شوند.

تابع **select_survivors** وظیفه انتخاب اعضای باقی‌مانده برای نسل بعدی در الگوریتم ژنتیک را دارد. این تابع با مرتب‌سازی افراد بر اساس مقدار فیتنسشان، اعضایی با بهترین فیتنس را انتخاب می‌کند و آنها را در لیست بازمی‌گرداند.

مراحل اجرای تابع به شرح زیر است:

لیست افراد جمع‌آوری می‌شود . این مرحله با استفاده از تابع **sorted** اقدام به مرتب‌سازی افراد بر اساس مقدار فیتنسشان می‌نماید.

تعداد اعضای مورد نیاز برای تشکیل نسل بعدی انتخاب می‌شود. در اینجا، تعداد اعضای جمعیت مساوی با اندازه جمعیت اولیه استفاده شده است . اعضای مورد نیاز از ابتدای لیست مرتب‌شده (که فیتنس کمترین را دارند) انتخاب می‌شوند.

لیست اعضای باقی‌مانده به عنوان خروجی تابع بازمی‌گردانده می‌شود .

تابع **select_survivors** در واقع وظیفه حفظ اعضای با کیفیت تر برای تولید نسل بعدی را دارد و از فرآیند انتخاب آن‌ها بر اساس فیتنس بهره می‌برد.

تابع **select_survivors1** نیز همانند **select_survivors** وظیفه انتخاب اعضای باقی‌مانده برای نسل بعدی در الگوریتم ژنتیک را دارد. با این حال، روش انتخاب این اعضا در این تابع تفاوت دارد. در **select_survivors1** از روش تورنمنت (Tournament Selection) استفاده شده است. در هر مرحله، ۳ افراد به صورت تصادفی از جمعیت انتخاب می‌شوند و از بین آن‌ها، کسی که فیتنس کمتری داشته باشد به عنوان برنده انتخاب می‌شود و به جمعیت بعدی افزوده می‌شود .

مقدار ۳ (تعداد افراد در هر تورنمنت) و روش انتخاب برنده (کسانی که فیتنس کمتری دارند) قابل تنظیم هستند و می‌توانند با تغییرات در کد تغییر یابند. برخلاف **select_survivors** که از مرتب‌سازی استفاده می‌کند و اعضای با بهترین فیتنس را انتخاب می‌کند، **select_survivors1** به صورت تصادفی انتخاب اعضایی با فیتنس کمتر را انجام می‌دهد.

فرق اصلی میان این دو تابع در روش انتخاب اعضا و در نهایت، افرادی که در نسل بعدی باقی میمانند.

تابع **run_genetic_algorithm** اجرای اصلی الگوریتم ژنتیک بر روی مسئله بهینه‌سازی مسیر را انجام می‌دهد. این تابع به صورت خاص الگوریتم ژنتیک را برای بهینه‌سازی مسیر در یک نقشه جاده اجرا می‌کند. در زیر توضیحاتی در مورد اجزای اصلی این تابع آورده شده‌اند:

1.تولید جمعیت اولیه (generate_population): ابتدا یک جمعیت اولیه از مسیرهای ممکن تولید می‌شود. هر مسیر به عنوان یک فرد در جمعیت نمایان می‌شود.

2. حلقه اصلی الگوریتم :

- **انتخاب والدین (select_parents):** از جمعیت فعلی، دو والدین به صورت تورنمنت انتخاب می‌شوند. این تابع در دو نسخه انتخاب والدین انجام می‌دهد: **select_parents** و **select_parents1**.
- **تولید فرزندان (crossover و crossover1):** با استفاده از والدین انتخاب‌شده، فرزندان جدید با استفاده از عملیات ترکیب (crossover) تولید می‌شوند. این تابع در دو نسخه انتخاب ترکیب انجام می‌دهد: **crossover** و **crossover1** .
- **میوتیشن (mutate و mutate1):** احتمال میوتیشن روی فرزندان اعمال می‌شود. میوتیشن ممکن است مسیرهای انتخاب شده را تغییر دهد.
- **انتخاب بازماندگان (select_survivors و select_survivors1):** از میان والدین و فرزندان تولید شده، اعضایی که باید در نسل بعدی باقی بمانند با استفاده از یک معیار انتخاب انتخاب می‌شوند. این تابع در دو نسخه انتخاب بازماندگان انجام می‌دهد : **select_survivors** و **select_survivors1**

این فرآیند تکرار می‌شود تا به تعداد دوره‌های تعیین شده برسد (max generation)

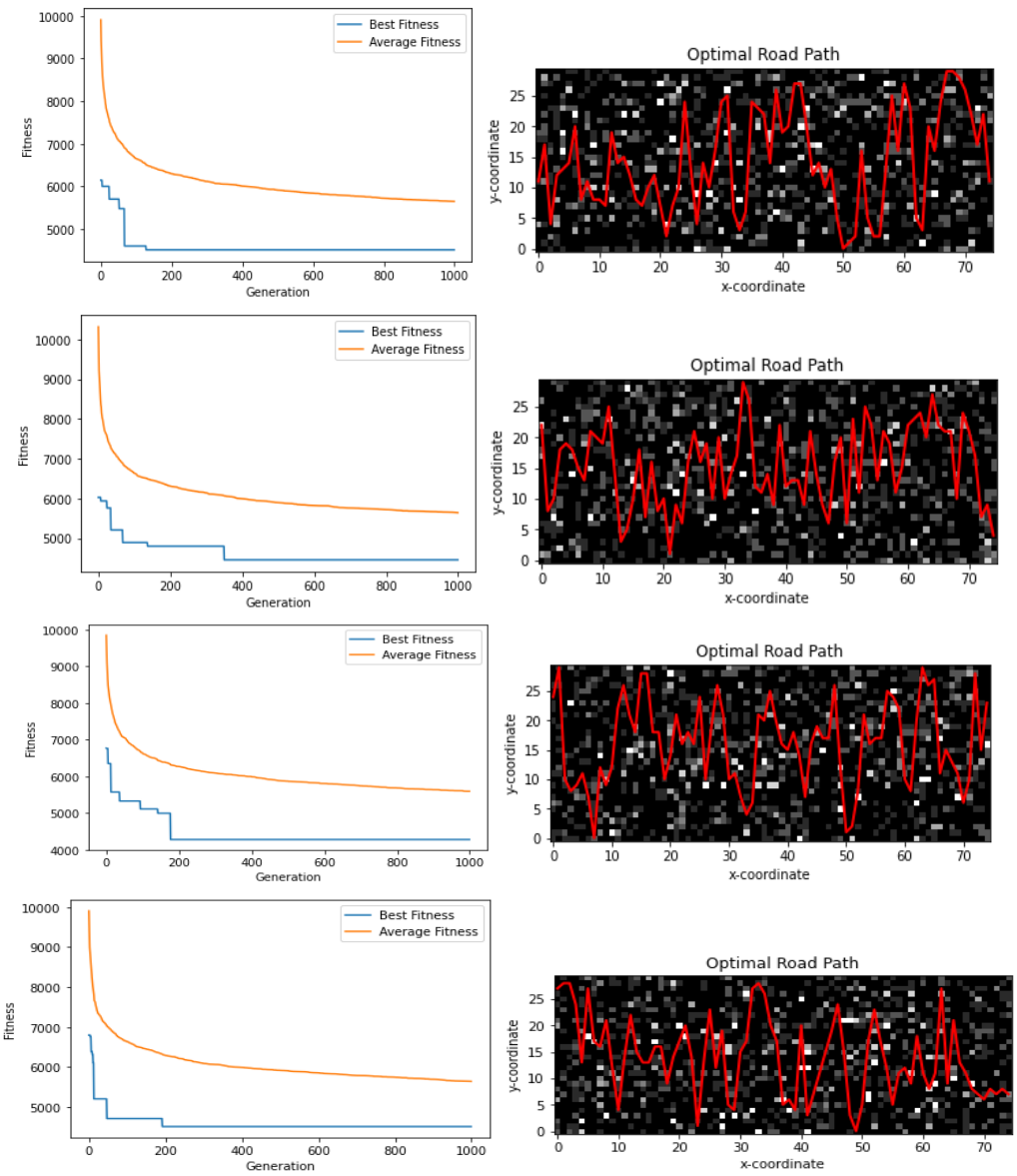
نمایش نتایج :

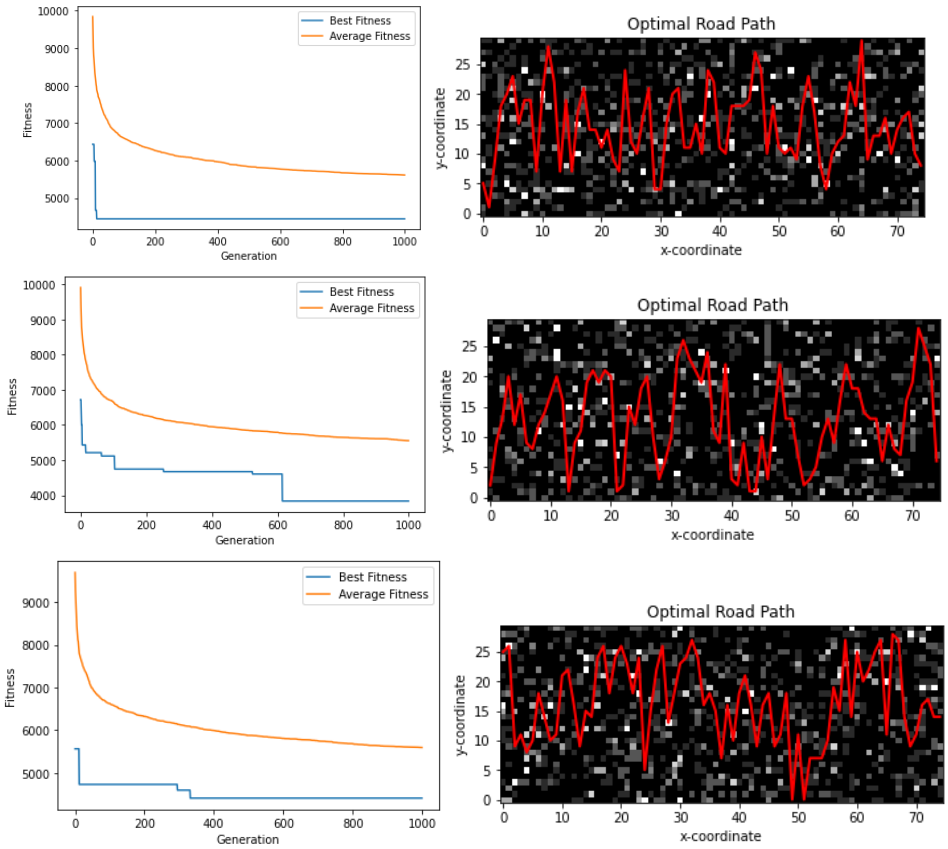
مسیر بهینه یافت شده به همراه نقشه جاده و تغییرات در ارزش فیتنس در هر نسل نمایش داده می‌شوند.

در ادامه همانطور که از ما خواسته شد تا اپراتور های دیگه را پیاده سازی کنیم و آن ها را از نظر فیتنس و پیدا کرد مسیر بهینه در این پروژه با هم مقایسه کنیم ' این کار را با سه اپراتور که در بالا توضیح داده شد انجام دادیم و خروجی به شکل زیر است :

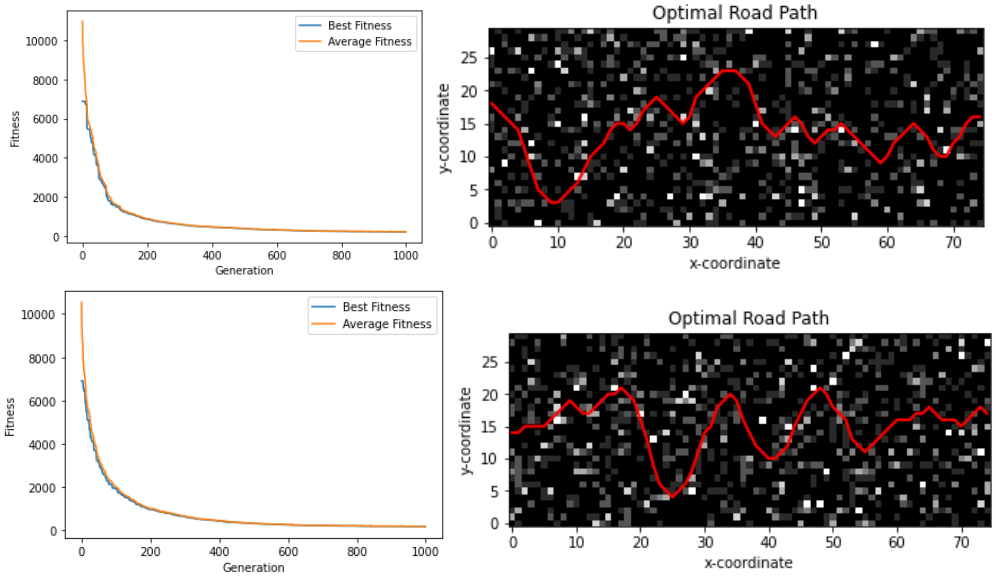
نکته : برای مقایسه هر فیتنس همانطور که از ما خواسته شد ما ۳ اپراتور را با هم دیگه مقایسه کردیم و خروجی را به شکل زیر نمایش میدم. در این خروجی ها میزان فیتنس و میانگین آن برای تاپل داده شده به نمایش گذاشته شده است :

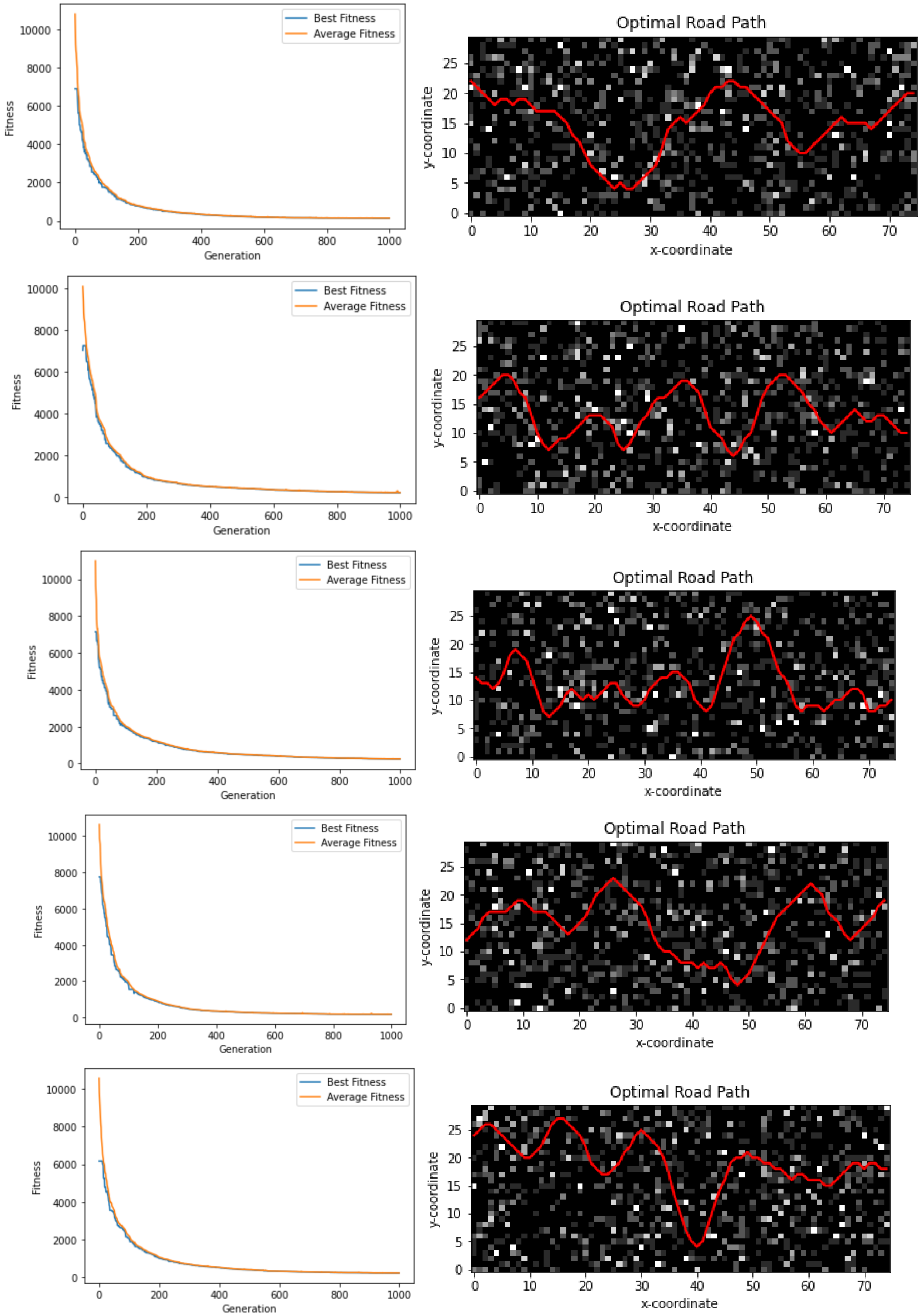
Setup 1: crossover , mutation, selectsurvivors



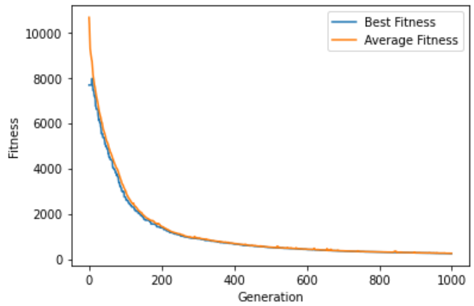


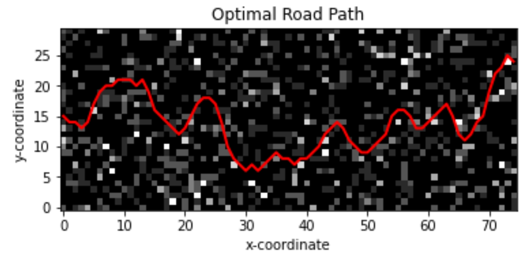
Setup 2 : crossover 1, mutation1, select survivors1

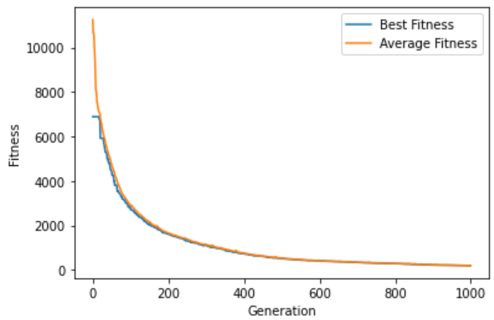


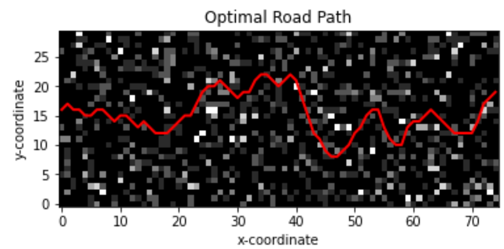

Best Fitness
Average Fitness
Fitness
Generation
Optimal Road Path
y-coordinate
x-coordinate

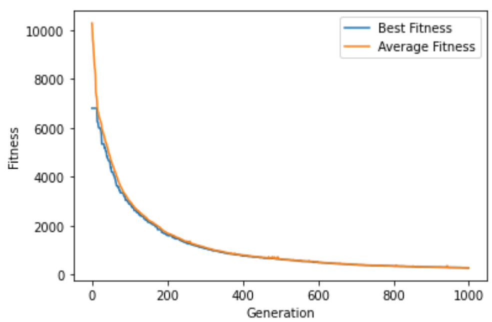
Setup 3: crossover, mutation1, select survivors1

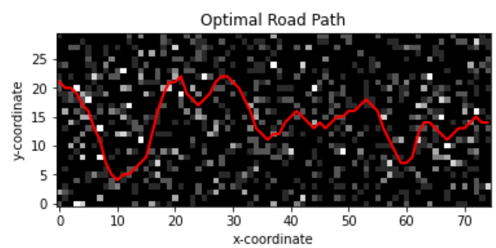


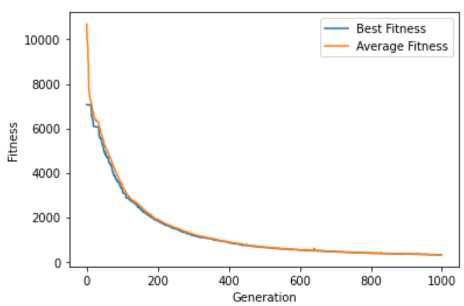


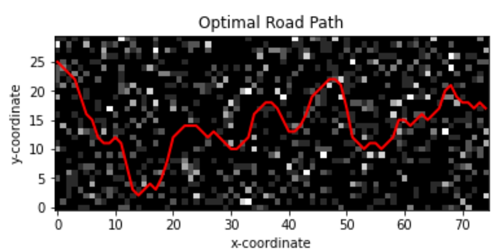


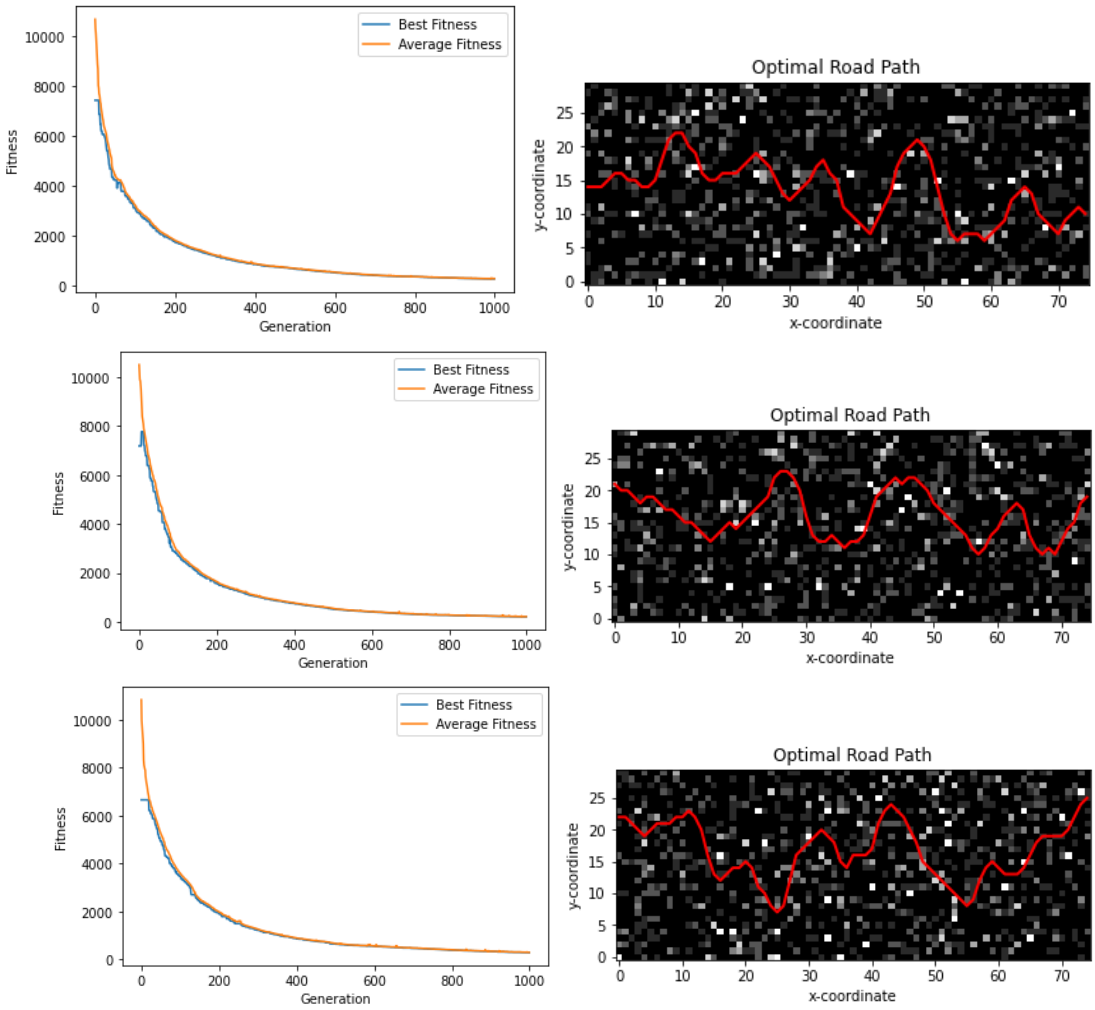




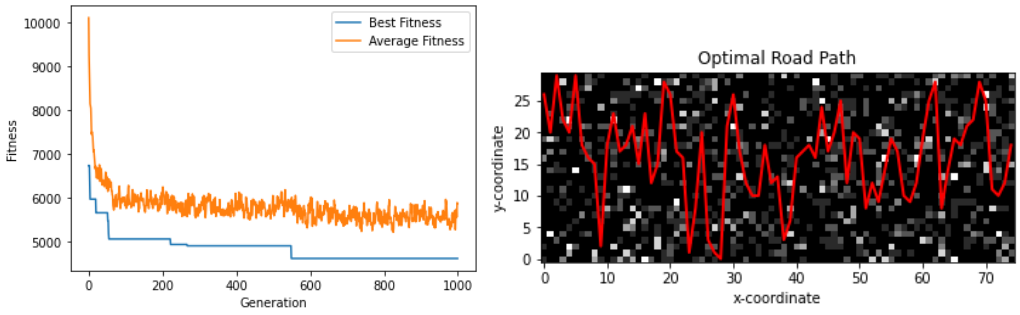


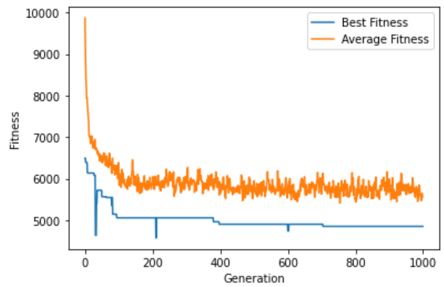


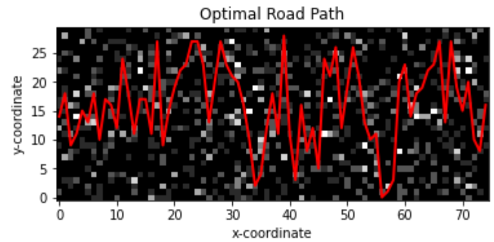


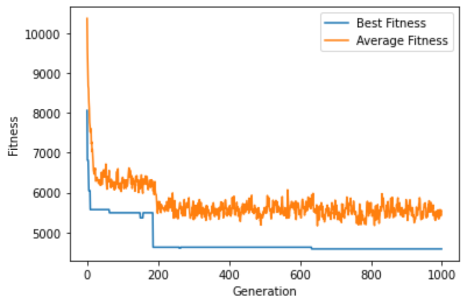


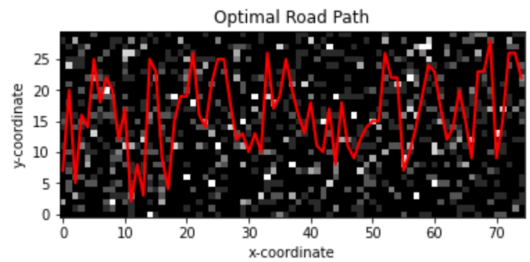
Setup 4: crossover ,mutation, select survivors1

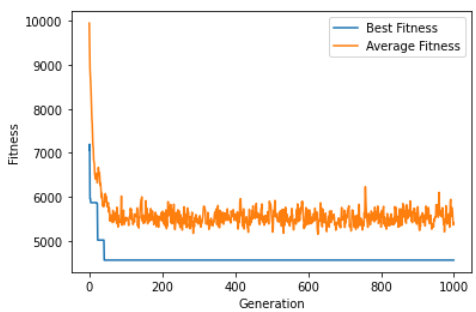


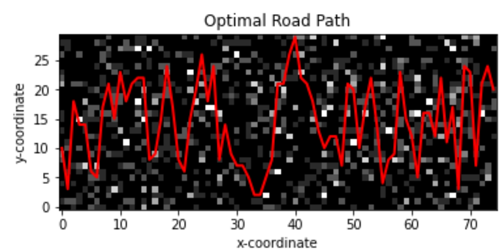

Best Fitness
Average Fitness
Fitness
Generation

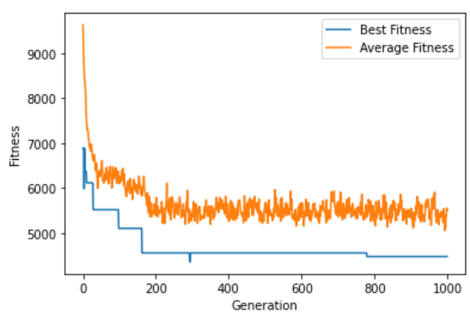

Optimal Road Path
y-coordinate
x-coordinate

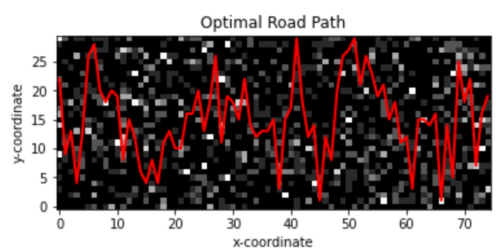

Best Fitness
Average Fitness
Fitness
Generation

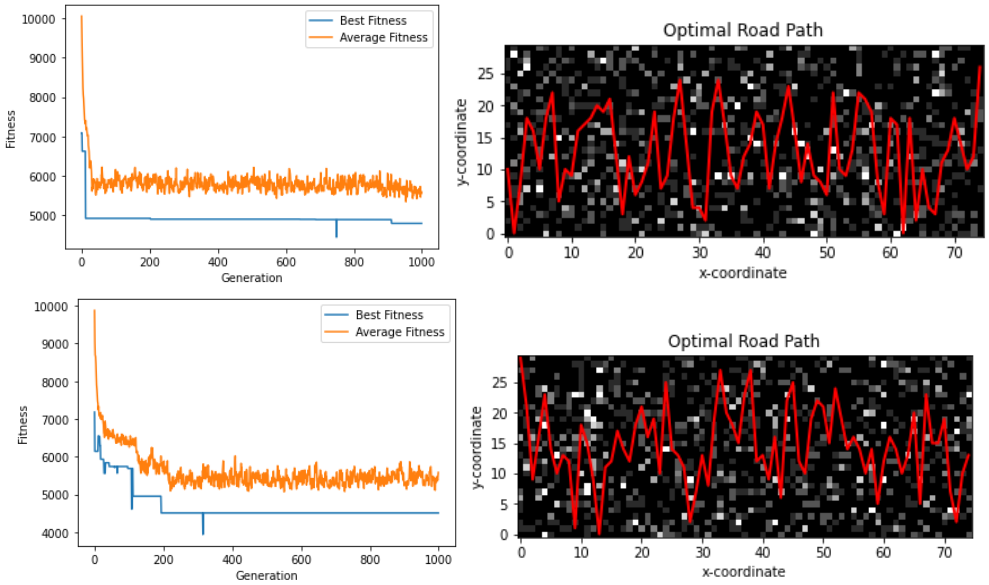

Optimal Road Path
y-coordinate
x-coordinate


Best Fitness
Average Fitness
Fitness
Generation

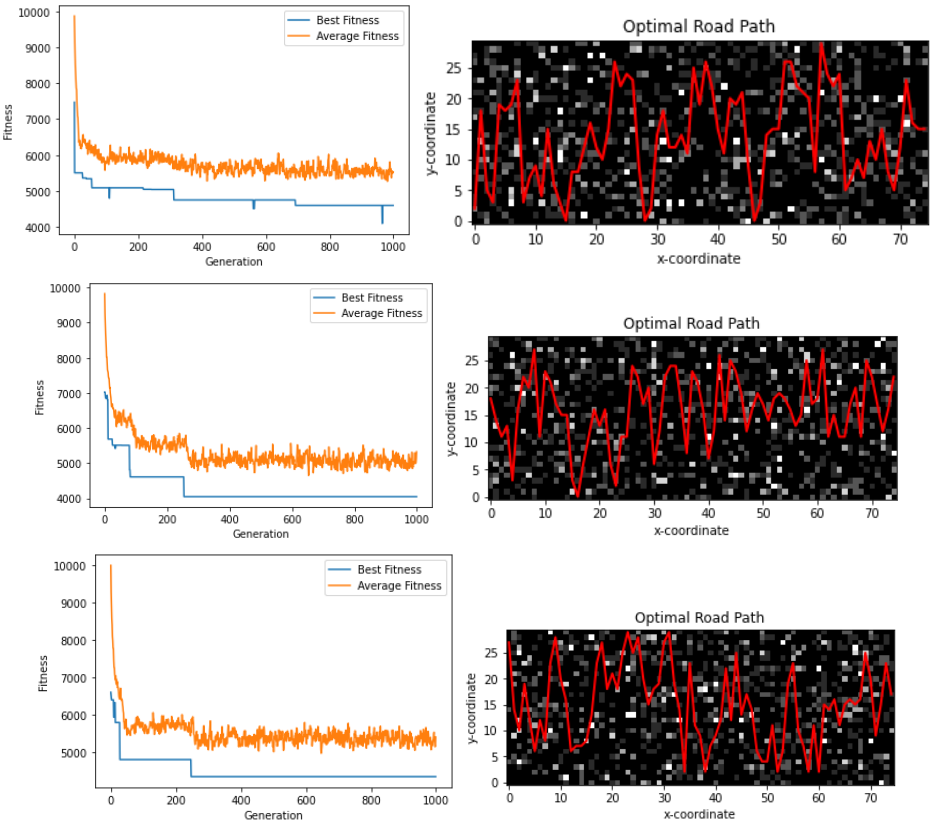

Optimal Road Path
y-coordinate
x-coordinate

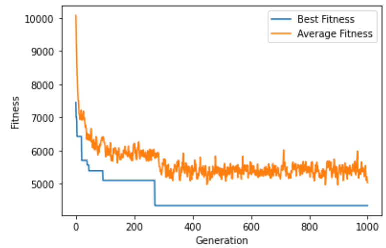

Best Fitness
Average Fitness
Fitness
Generation

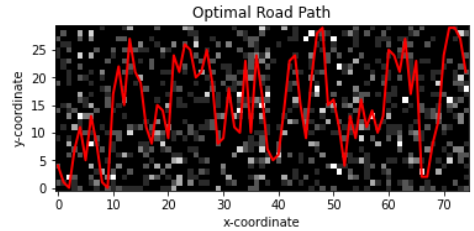

Optimal Road Path
y-coordinate
x-coordinate

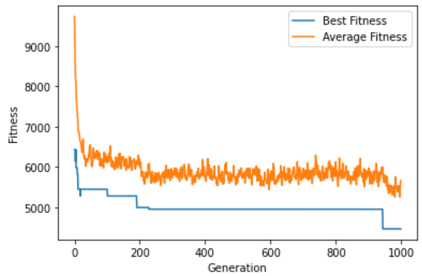


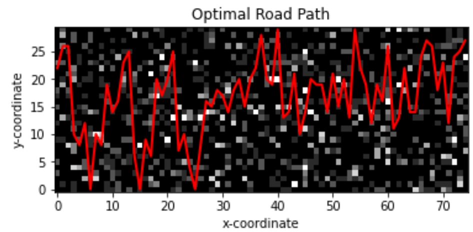
Setup 5: crossover 1, mutation, select survivors1

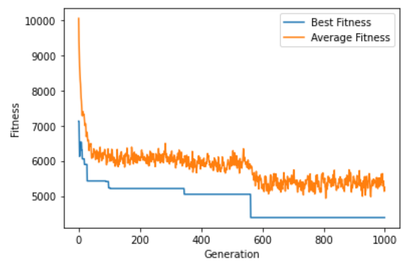


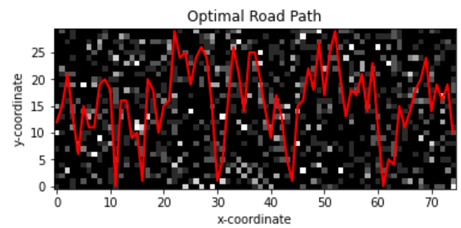


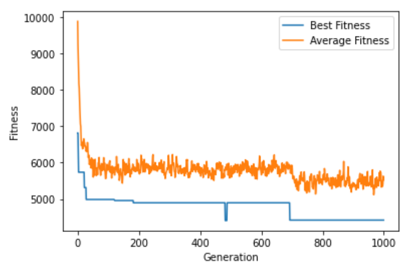


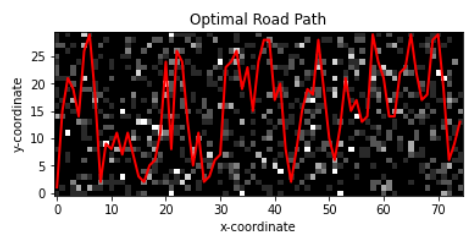




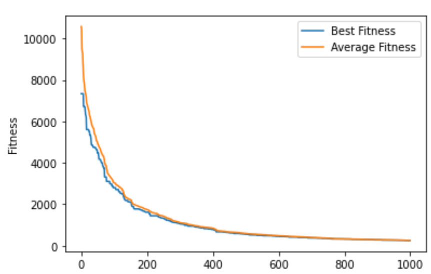


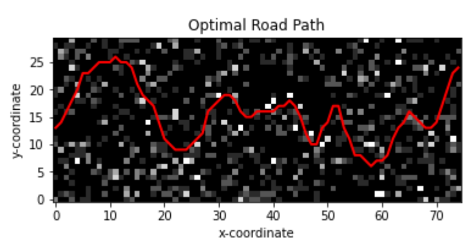


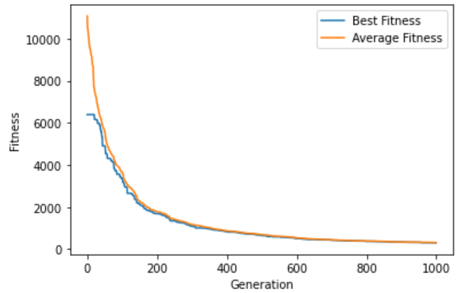
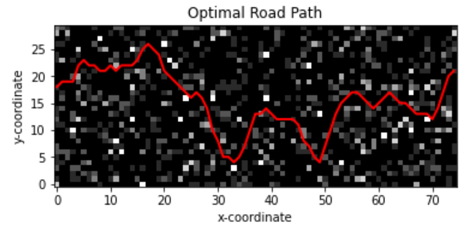
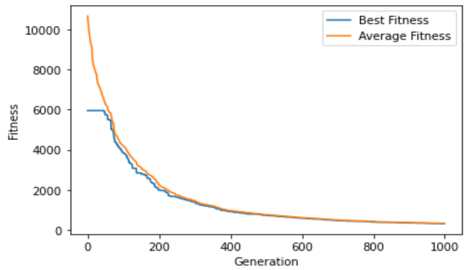
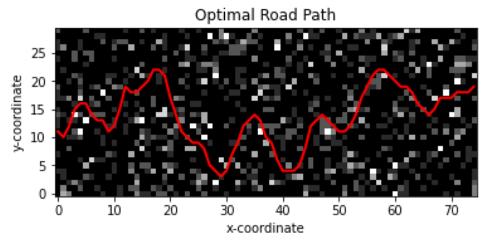
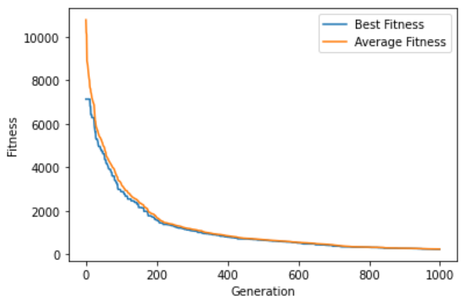
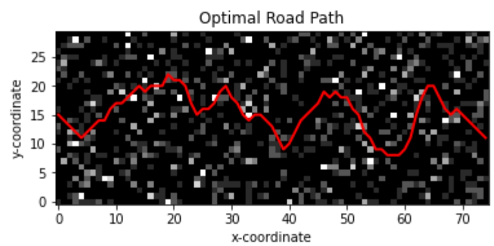
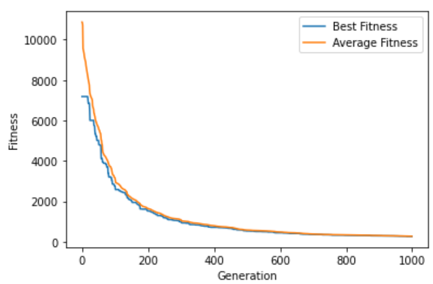
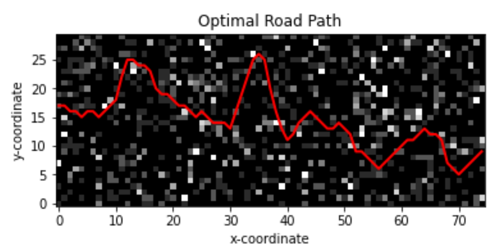
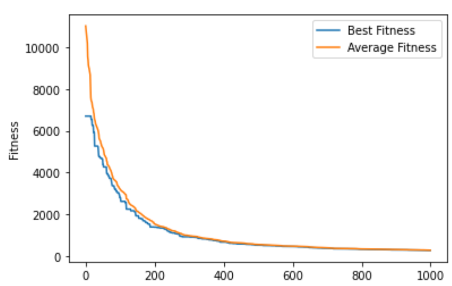
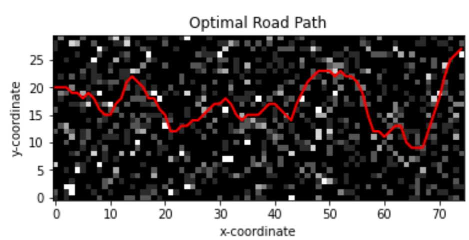




Setup 6 : crossover 1, mutation1, select survivors

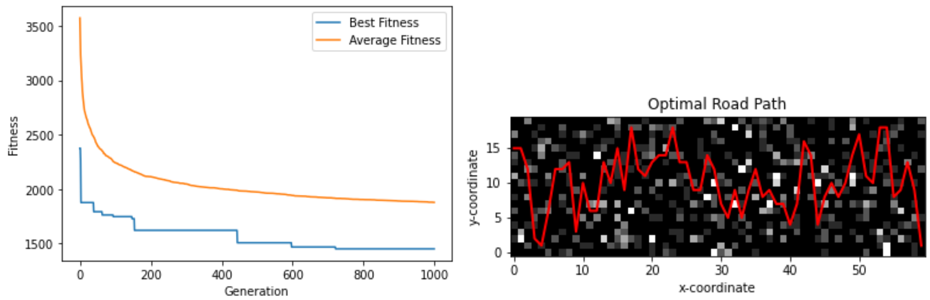


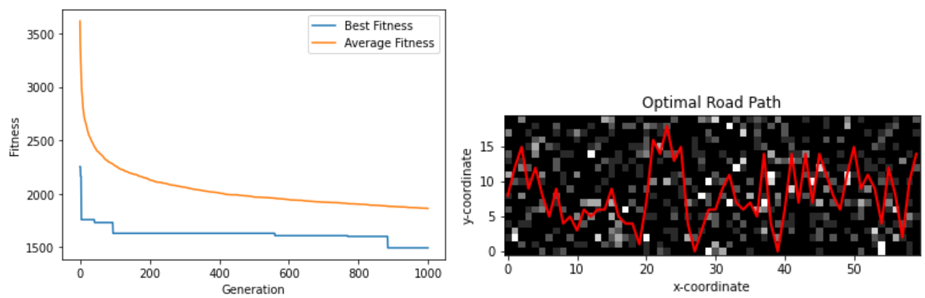


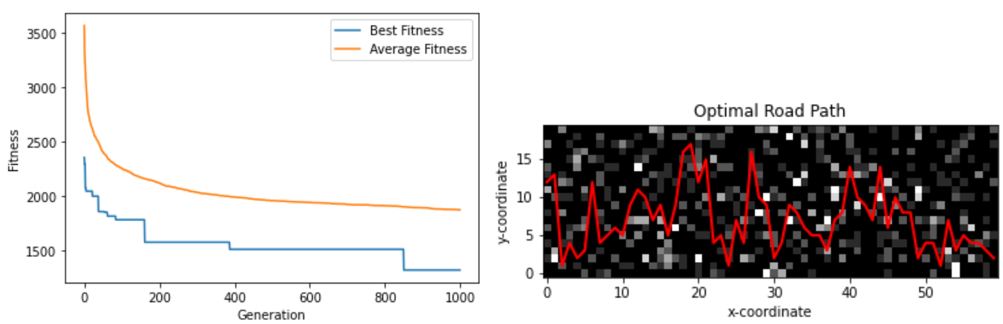

Best Fitness
Average Fitness
Fitness
Generation
0 2000 4000 6000 8000 10000
0 200 400 600 800 1000

Optimal Road Path
y-coordinate
x-coordinate
0 5 10 15 20 25
0 10 20 30 40 50 60 70

Best Fitness
Average Fitness
Fitness
Generation
0 2000 4000 6000 8000 10000
0 200 400 600 800 1000

Optimal Road Path
y-coordinate
x-coordinate
0 5 10 15 20 25
0 10 20 30 40 50 60 70

Best Fitness
Average Fitness
Fitness
Generation
0 2000 4000 6000 8000 10000
0 200 400 600 800 1000

Optimal Road Path
y-coordinate
x-coordinate
0 5 10 15 20 25
0 10 20 30 40 50 60 70

Best Fitness
Average Fitness
Fitness
Generation
0 2000 4000 6000 8000 10000
0 200 400 600 800 1000

Optimal Road Path
y-coordinate
x-coordinate
0 5 10 15 20 25
0 10 20 30 40 50 60 70

Best Fitness
Average Fitness
Fitness
Generation
0 2000 4000 6000 8000 10000
0 200 400 600 800 1000

Optimal Road Path
y-coordinate
x-coordinate
0 5 10 15 20 25
0 10 20 30 40 50 60 70

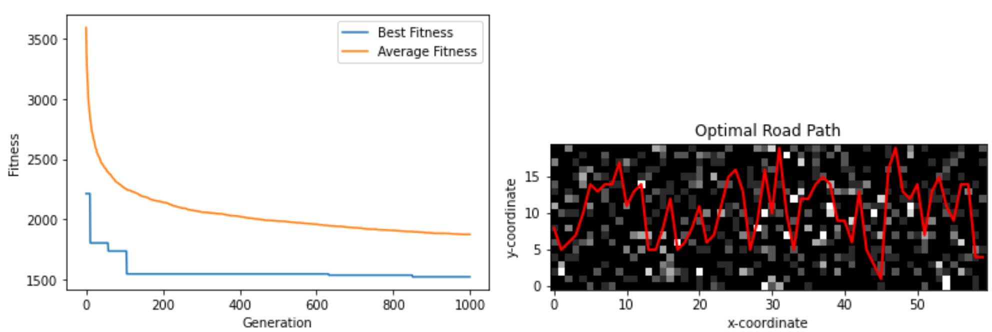
در این خروجی ها میزان فیتنس و میانگین آن برای فایل map داده شده به نمایش گذاشته شده است :

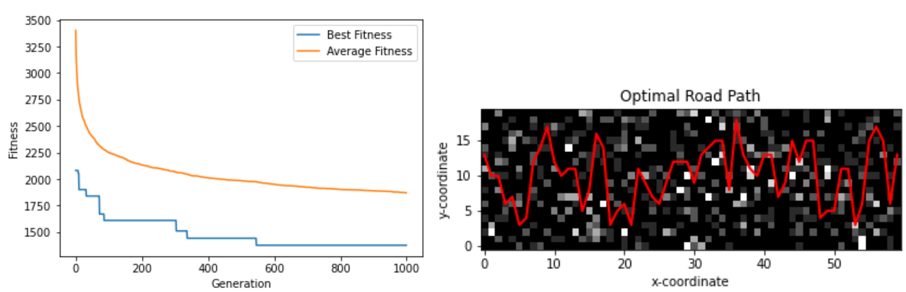
Setup 1 : crossover, mutation, select survivors

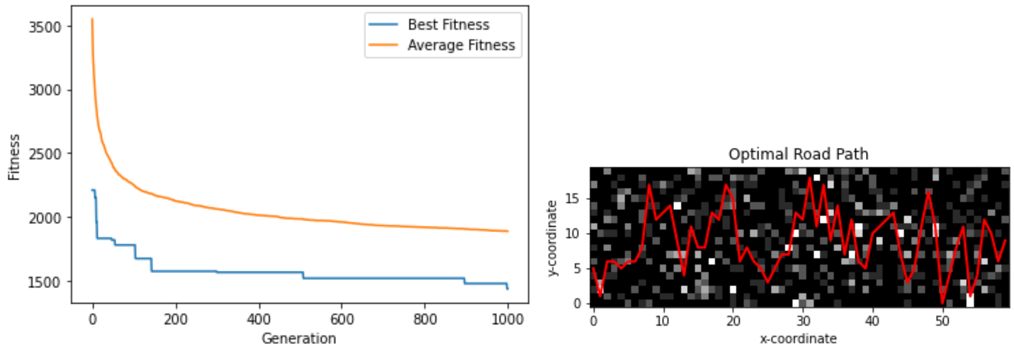


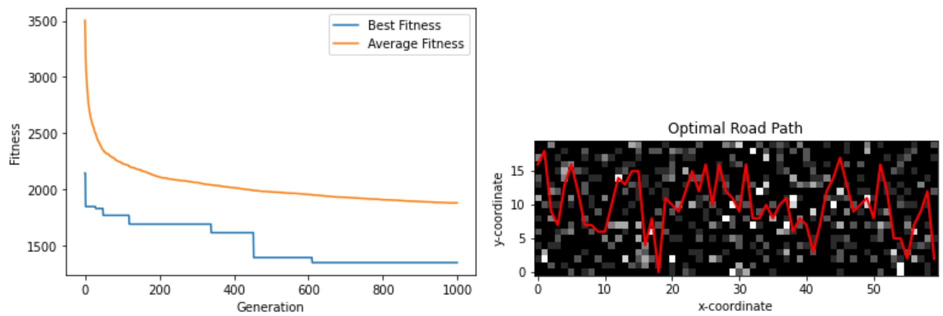




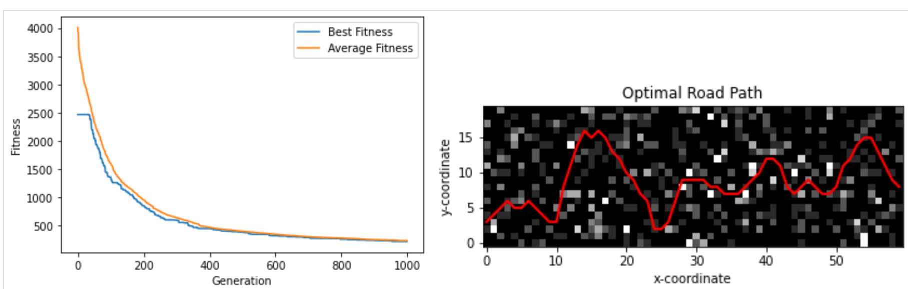


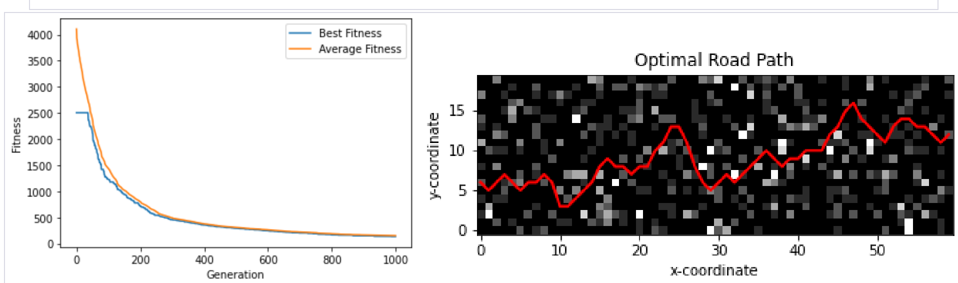


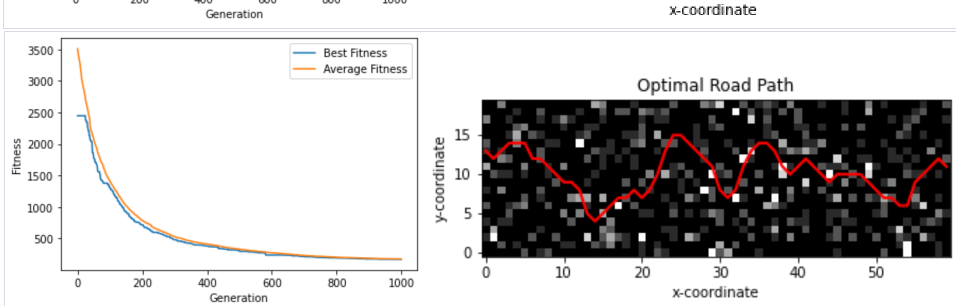


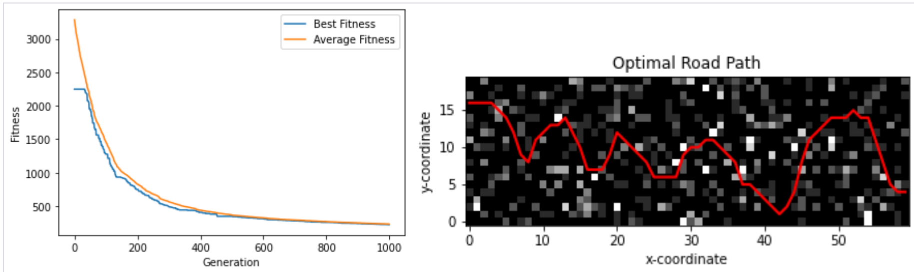


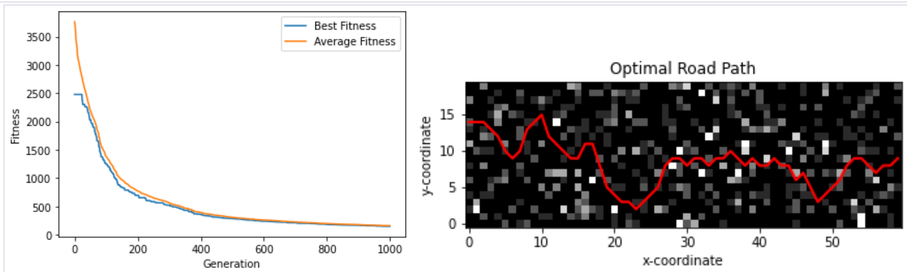
Setup 2 : crossover, mutation1, select survivors

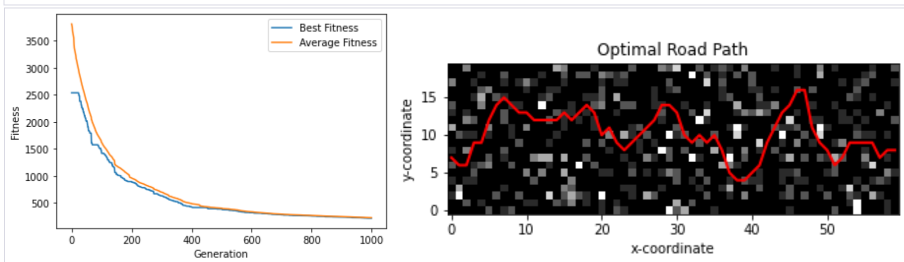


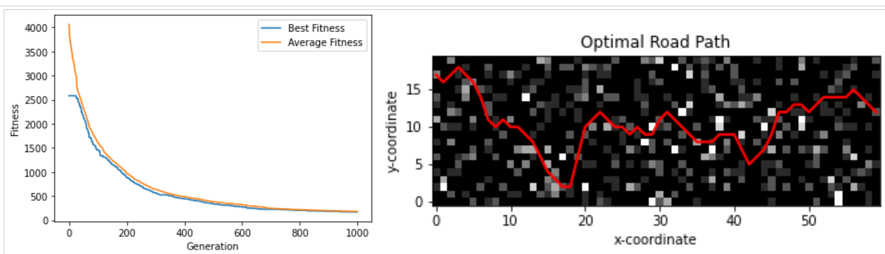




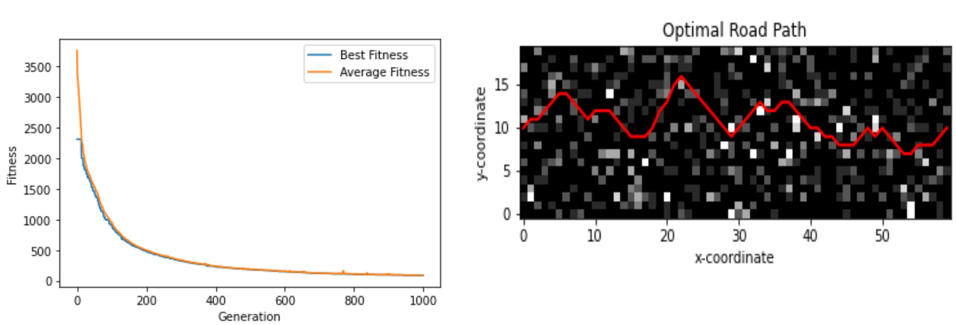


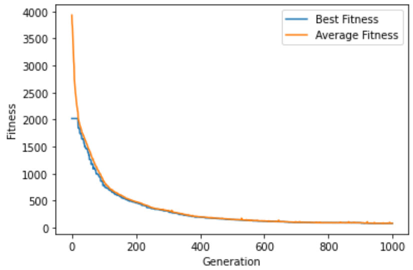
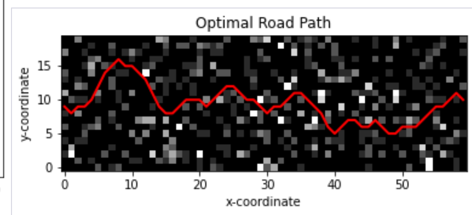
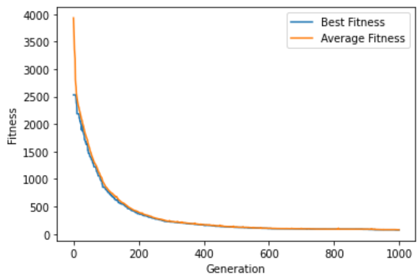
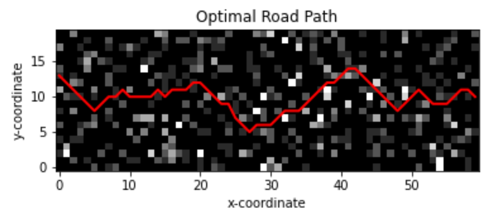
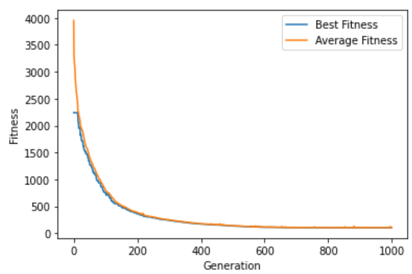
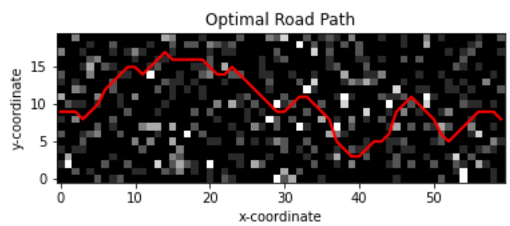
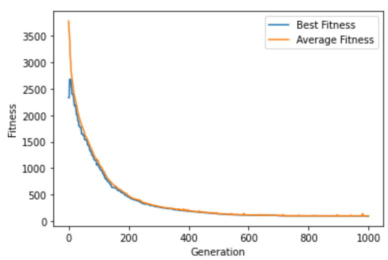
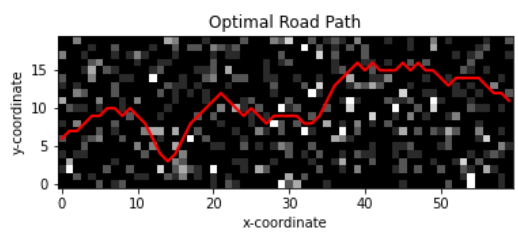
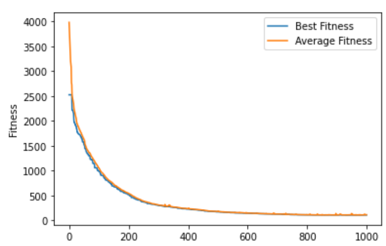
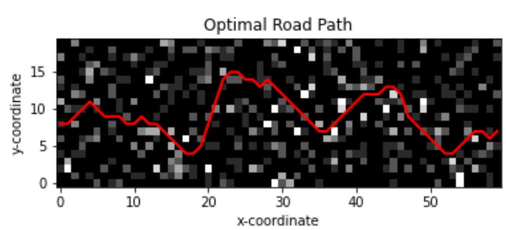
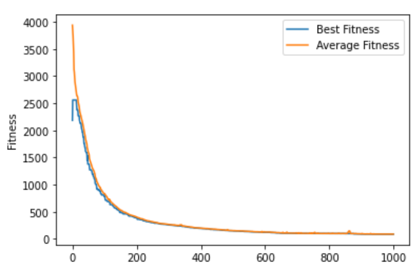
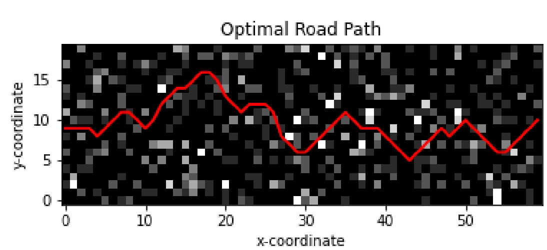




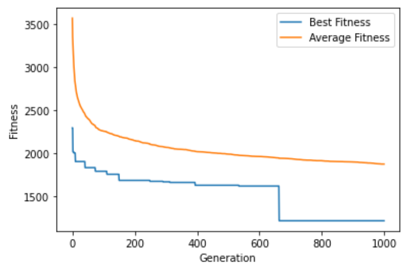


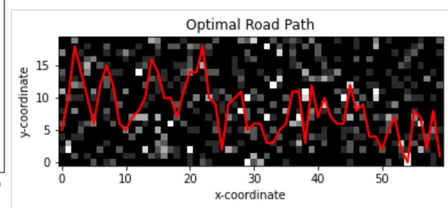
Setup 3 : crossover, mutation1, select survivors1

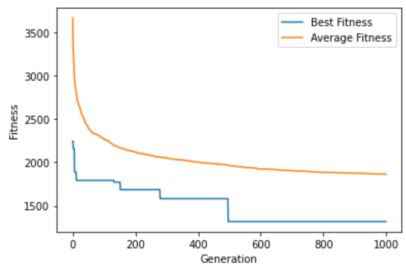


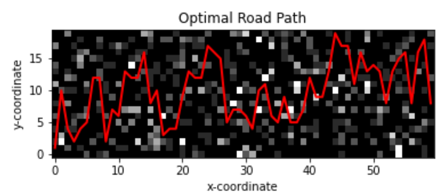

Best Fitness
Average Fitness
Fitness
Generation

Optimal Road Path
y-coordinate
x-coordinate

Best Fitness
Average Fitness
Fitness
Generation

Optimal Road Path
y-coordinate
x-coordinate

Best Fitness
Average Fitness
Fitness
Generation

Optimal Road Path
y-coordinate
x-coordinate

Best Fitness
Average Fitness
Fitness
Generation

Optimal Road Path
y-coordinate
x-coordinate

Best Fitness
Average Fitness
Fitness
Generation

Optimal Road Path
y-coordinate
x-coordinate

Best Fitness
Average Fitness
Fitness
Generation

Optimal Road Path
y-coordinate
x-coordinate

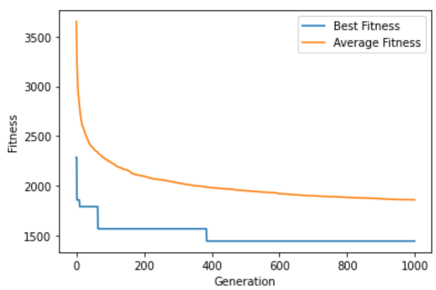
Setup 4 : crossover 1, mutation, select survivors

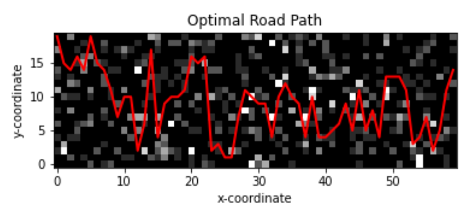


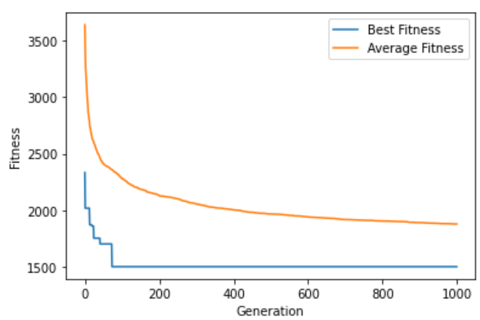


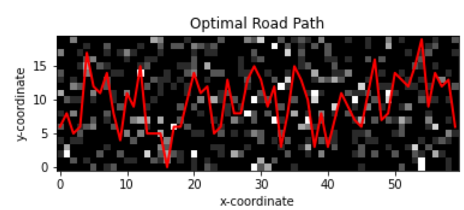


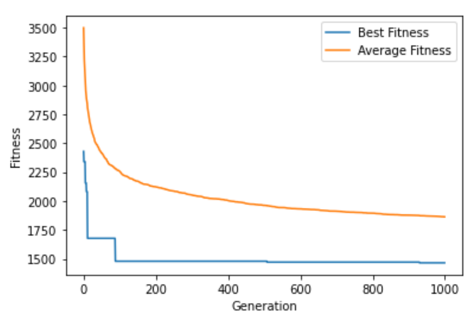


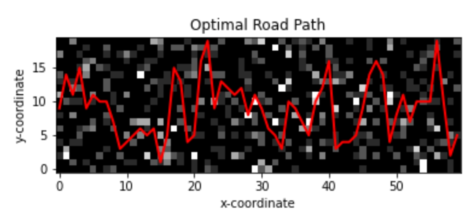


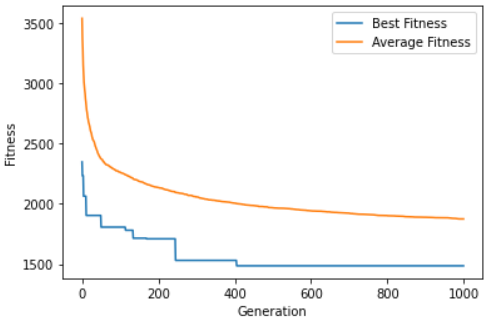


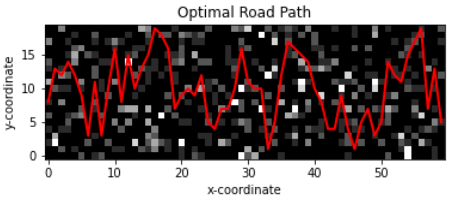


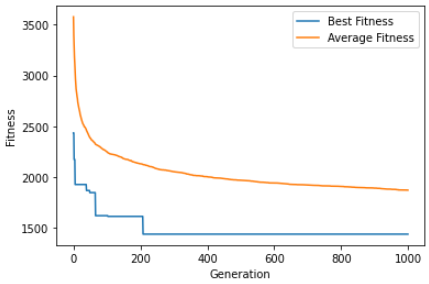


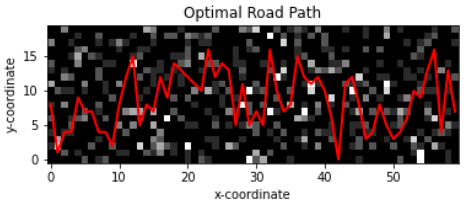





Best Fitness
Average Fitness
Fitness
Generation
3500
3000
2500
2000
1500
0
200
400
600
800
1000


Optimal Road Path
y-coordinate
x-coordinate
0
5
10
15
0
10
20
30
40
50


Best Fitness
Average Fitness
Fitness
Generation
3500
3000
2500
2000
1500
0
200
400
600
800
1000


Optimal Road Path
y-coordinate
x-coordinate
0
5
10
15
0
10
20
30
40
50