Assignment #HW1 – Genetic Algorithm
CSSE490: Bio-Inspired Artificial Intelligence
Fall 2022

Prepared and submitted by:
Zach Decker and Andrea Chen

Collaboration and resources:
We worked as a group.

Resources I used to complete this assignment (websites, textbook, friends, etc.):
N/A

Checkpoints

For each checkpoint, simply include a screenshot of the output of your program demonstrating that the code is doing what it is supposed to.

All the checkpoints were created with seed 7431. In our model, the lower the fitness is, the better the individual.

Check Point 1.

```
9,6,11,5,12,2,3,15,8,4,14,10,13,1,7, fitness: 0.0
14,8,2,6,12,11,4,1,3,15,5,7,13,10,9, fitness: 0.0
5,15,7,2,6,10,3,8,14,4,13,1,12,9,11, fitness: 0.0
11,9,1,14,3,6,8,2,4,7,15,12,13,5,10, fitness: 0.0
12,5,10,4,9,15,8,6,11,14,3,1,13,7,2, fitness: 0.0
10,7,1,15,13,11,6,4,9,3,14,8,2,5,12, fitness: 0.0
3,5,12,10,9,11,8,2,15,14,13,4,7,1,6, fitness: 0.0
10,2,12,5,13,11,4,15,14,8,7,3,9,6,1, fitness: 0.0
12,3,15,2,1,10,11,7,6,9,14,13,5,4,8, fitness: 0.0
3,9,11,12,5,8,6,4,10,1,2,14,15,7,13, fitness: 0.0
```

Initial population(#1 trial)

```
9,6,11,5,12,2,3,15,8,4,14,10,13,1,7, fitness: 0.0
14,8,2,6,12,11,4,1,3,15,5,7,13,10,9, fitness: 0.0
5,15,7,2,6,10,3,8,14,4,13,1,12,9,11, fitness: 0.0
11,9,1,14,3,6,8,2,4,7,15,12,13,5,10, fitness: 0.0
12,5,10,4,9,15,8,6,11,14,3,1,13,7,2, fitness: 0.0
10,7,1,15,13,11,6,4,9,3,14,8,2,5,12, fitness: 0.0
3,5,12,10,9,11,8,2,15,14,13,4,7,1,6, fitness: 0.0
10,2,12,5,13,11,4,15,14,8,7,3,9,6,1, fitness: 0.0
12,3,15,2,1,10,11,7,6,9,14,13,5,4,8, fitness: 0.0
3,9,11,12,5,8,6,4,10,1,2,14,15,7,13, fitness: 0.0
```

Initial population(#2 trial)

Check Point 2.

```
9,6,11,5,12,2,3,15,8,4,14,10,13,1,7, fitness: 0.0
14,8,2,6,12,11,4,1,3,15,5,7,13,10,9, fitness: 0.0
5,15,7,2,6,10,3,8,14,4,13,1,12,9,11, fitness: 0.0
11,9,1,14,3,6,8,2,4,7,15,12,13,5,10, fitness: 0.0
12,5,10,4,9,15,8,6,11,14,3,1,13,7,2, fitness: 0.0
10,7,1,15,13,11,6,4,9,3,14,8,2,5,12, fitness: 0.0
3,5,12,10,9,11,8,2,15,14,13,4,7,1,6, fitness: 0.0
10,2,12,5,13,11,4,15,14,8,7,3,9,6,1, fitness: 0.0
12,3,15,2,1,10,11,7,6,9,14,13,5,4,8, fitness: 0.0
3,9,11,12,5,8,6,4,10,1,2,14,15,7,13, fitness: 0.0
-------Above is initial population--------
9,6,11,5,12,2,3,15,8,4,14,10,13,1,7, fitness: 675.8481975940309
14,8,2,6,12,11,4,1,3,15,5,7,13,10,9, fitness: 616.1892612819672
5,15,7,2,6,10,3,8,14,4,13,1,12,9,11, fitness: 655.7493314848725
11,9,1,14,3,6,8,2,4,7,15,12,13,5,10, fitness: 725.1561273464084
12,5,10,4,9,15,8,6,11,14,3,1,13,7,2, fitness: 650.3499484883732
10,7,1,15,13,11,6,4,9,3,14,8,2,5,12, fitness: 566.9747947208963
3,5,12,10,9,11,8,2,15,14,13,4,7,1,6, fitness: 718.6269936981212
10,2,12,5,13,11,4,15,14,8,7,3,9,6,1, fitness: 670.1378095828576
12,3,15,2,1,10,11,7,6,9,14,13,5,4,8, fitness: 724.3166010839238
3,9,11,12,5,8,6,4,10,1,2,14,15,7,13, fitness: 700.6243666129802
--------Above is initial population with fitness calculated----------
```

Initial population with fitness

Check Point 3.

```
10,7,1,15,13,11,6,4,9,3,14,8,2,5,12, fitness: 566.9747947208963
14,8,2,6,12,11,4,1,3,15,5,7,13,10,9, fitness: 616.1892612819672
12,5,10,4,9,15,8,6,11,14,3,1,13,7,2, fitness: 650.3499484883732
5,15,7,2,6,10,3,8,14,4,13,1,12,9,11, fitness: 655.7493314848725
10,2,12,5,13,11,4,15,14,8,7,3,9,6,1, fitness: 670.1378095828576
9,6,11,5,12,2,3,15,8,4,14,10,13,1,7, fitness: 675.8481975940309
3,9,11,12,5,8,6,4,10,1,2,14,15,7,13, fitness: 700.6243666129802
3,5,12,10,9,11,8,2,15,14,13,4,7,1,6, fitness: 718.6269936981212
12,3,15,2,1,10,11,7,6,9,14,13,5,4,8, fitness: 724.3166010839238
11,9,1,14,3,6,8,2,4,7,15,12,13,5,10, fitness: 725.1561273464084
-------Above is sorted population--------
```

Sort population from low fitness to high

Check Point 4.

```
parent:        10,7,1,15,13,11,6,4,9,3,14,8,2,5,12, fitness: 566.9747947208963
mutated child: 10,7,13,15,1,11,6,4,9,3,14,8,2,5,12, fitness: 0.0
```

Parent and its mutated child (Swap mutation)

Check Point 5.

```
parent1:10,7,1,15,13,11,6,4,9,3,14,8,2,5,12, fitness: 566.9747947208963
parent2:14,8,2,6,12,11,4,1,3,15,5,7,13,10,9, fitness: 616.1892612819672
child:   1,15,7,13,10,11,6,4,9,3,14,8,2,5,12, fitness: 0.0
```

With crossover

Check Point 6.

```
Best Fitness: 566.9747947208963 at generation 0
Best Fitness: 561.5690631654855 at generation 2
Best Fitness: 550.6194530097625 at generation 3
Best Fitness: 512.7229784677396 at generation 12
Best Fitness: 454.15691898202203 at generation 15
Best Fitness: 448.72301512781456 at generation 40
Best Fitness: 444.81109096336206 at generation 48
Best Fitness: 439.4494482174292 at generation 50
Best Fitness: 439.44944821742916 at generation 53
Best Fitness: 435.3661366010024 at generation 61
Best Fitness: 420.7803847882475 at generation 63
Best Fitness: 420.78038478824743 at generation 73
Best Fitness: 418.4106294670072 at generation 88
```

Best fitness whenever it's historically highest

Check Point 7.

```
Best Fitness: 566.9747947208963 at generation 0
Best Fitness: 566.9747947208963 at generation 1
Best Fitness: 561.5690631654855 at generation 2
Best Fitness: 550.6194530097625 at generation 3
Best Fitness: 550.6194530097625 at generation 4
Best Fitness: 550.6194530097625 at generation 5
Best Fitness: 550.6194530097625 at generation 6
Best Fitness: 550.6194530097625 at generation 7
Best Fitness: 550.6194530097625 at generation 8
Best Fitness: 550.6194530097625 at generation 9
Best Fitness: 550.6194530097625 at generation 10
```

With elitism=1, the best fitness never goes down

Check Point 8.

```
Best Fitness: 566.9747947208963 at generation 0
Best Fitness: 566.9747947208963 at generation 1
Best Fitness: 566.9747947208963 at generation 2
Best Fitness: 566.9747947208963 at generation 3
Best Fitness: 566.9747947208963 at generation 4
Best Fitness: 566.9747947208963 at generation 5
Best Fitness: 566.9747947208963 at generation 6
Best Fitness: 566.9747947208963 at generation 7
Best Fitness: 566.9747947208963 at generation 8
Best Fitness: 566.9747947208963 at generation 9
Best Fitness: 566.9747947208963 at generation 10
```

Without crossover but with elitism

```
Best Fitness: 566.9747947208963 at generation 0
Best Fitness: 566.9747947208963 at generation 1
Best Fitness: 561.5690631654855 at generation 2
Best Fitness: 550.6194530097625 at generation 3
Best Fitness: 550.6194530097625 at generation 4
Best Fitness: 550.6194530097625 at generation 5
Best Fitness: 550.6194530097625 at generation 6
Best Fitness: 550.6194530097625 at generation 7
Best Fitness: 550.6194530097625 at generation 8
Best Fitness: 550.6194530097625 at generation 9
Best Fitness: 550.6194530097625 at generation 10
```

With crossover and elitism

Experiments

Instructions:

1. For some experiments a description is already provided to you, but you should amend or adjust as necessary (especially if you made any alterations). For your own experiments, you should describe what you are investigating and any specifics about the fitness function you are using.
2. Before you actually run an experiment, you should formulate a hypothesis. It is completely OK to make an error in your prediction, in fact, it is often when experiments produce unexpected results that we have the greatest opportunity to learn from them.
3. The parameters you used to run your experiment should be provided appropriately so that they can be easily reproduced by someone else. The use of a standardized configuration file makes it very easy and portable to allow someone else to attempt to reproduce the same results that you did. While you are free to use whatever format you want in your own codebase, we do request you fill it in the report so that it will be easy for us to copy and paste if we want to try to reproduce any results when grading.
4. For your results, you should provide **BOTH data and written observation**. The data can come in the form of one or more plots and tables of data. The observations should be objective statements about patterns or trends in the data you have provided.
5. After the results you should discuss the findings. This should be your attempt to explain the results, anything you learned from the experiment, any conclusions you feel you can make, and any speculations or ideas for additional experiments you might have.

Experiment #1

a. Description

Observe the average and variance in the number of generations required to evolve an approximate solution using only mutation. We use a dataset of a pre-solved traveling salesman problem with 15 cities on a 2D grid (link provided at the end of the experiment). The optimal solution is not always found, especially through mutation, so instead we will be finding the generation that created a solution within 15% of optimal, in one thousand generations. We chose this because the algorithm would rarely if ever make improvements past this point while still at most testing 1x10^-8 of the possible paths (in reality it would be much less than this).

b. Hypothesis

Something like this is hard to make a hypothesis, as we know little about the strength of this model. This being said, we hypothesize an average of about 120 with a median around 100. This is because over approximately 100 generations we believe the small population has usually converged and is only able to make incremental improvements, although we expect outliers.

c. Parameters

population.size=100
max.generations=1000
chromosome.length=15
mutation.rate=0.1/15
crossover.enable=false
elite.count=0
selection.method=truncation

d. Results

Table of Data Collected:

run #	1	2	3	4	5	6	7	8	9	10
Seed	*16781*	*50678*	*88264*	*88048*	*13207*	*12018*	*38840*	*98242*	*1288*	*31431*
Gens req.	*121*	*50*	*348*	*81*	*74*	*N/A*	*70*	*47*	*69*	*N/A*
Achieved Fitness	*323*	*325*	*319*	*323*	*321*	*N/A*	*322*	*321*	*321*	*N/A*

Observations:

Generations required (when reached):
- Average(mean): 107.5
- Median: 72.0
- Standard Deviation: 93.34211268232576

e. Discussion

Overall, our hypothesis was mainly supported. Only once in ten trials did the algorithm achieve this within 15% of optimal fitness past 150 generations and only twice past 100. These outliers are expected in any random

process such as a genetic algorithm. Our hypothesized numbers for average and median generations were too high, but relatively close to the actual result.

Experiment #2

a. Description

We want to investigate when the generation reaches the optimal path with only mutation with the set of cities from the last experiment.

b. Hypothesis

With only mutation, there is not a lot of diversity introduced, but the genome length is not very long either, so we'll expect it takes less than 200 generations for the population to reach best fitness (284.38).

c. Parameters

population.size=100
max.generations=1000
chromosome.length=15
mutation.rate=0.1/15
crossover.enable=false
elite.count=0
selection.method=truncation

d. Results

Table of Data Collected:

seed #	58662	37035	56965	3364	75201	3783	90419	33861	60355	91642
Gen. with highest fitness	*154*	*52*	*143*	*82*	*72*	*260*	*104*	*221*	*150*	*165*
Highest fitness	*315.03*	*357.42*	*284.38*	*330.56*	*312.24*	*329.71*	*308.16*	*284.38*	*284.38*	*335.25*

Observations:
Not all trails were able to reach the solution even with plenty of generations left. All improvements on the population happened during the first 300 generations. #37035 has the earliest stopping generation but the lowest fitness.

e. Discussion
With only mutation, some populations are able to gain the optimal solution while others cannot. But regardless of their success, the population's fitness seems to stabilize within 300 generations and no improvement is made afterwards. Our assumption is that the population's diversity becomes too low that everyone looks nearly identical, and mutation itself cannot provide enough randomness to the population to make it better. If the population cannot reach the solution before the diversity becomes too low, they'll lose the chance regardless of how much time is left. It'll be nicer if we had the diversity implemented so that we can test our assumption.

Experiment #3

a. Description

Observe the average and variance in the number of generations required to evolve an approximate solution using mutation and crossover. We use a dataset of a pre-solved traveling salesman problem with 15 cities on a 2D grid (link provided at the end of the experiment). The optimal solution is not always found, especially through mutation, so instead we will be finding the generation that created a solution within 10% of optimal, in ten thousand generations. We decided to test for 10% this time as we found it was consistently able to reach it well within the 10000 generation limit. We chose this because the algorithm would rarely make large improvements past this point while still at most testing .000001 of the possible paths (in reality it would be much less than this).

b. Hypothesis

Something like this is hard to make a hypothesis, as we know little about the strength of this model. This being said, we hypothesize an average of about 700 with a median around 500. This is because over approximately 800 generations we believe the small population of 100 has usually converged and is only able to make incremental improvements, although we expect outliers.

c. Parameters

population.size=100
max.generations=10000
chromosome.length=15
mutation.rate=0.1
crossover.enable=true
elite.count=0
selection.method=truncation

d. Results

Table of Data Collected:

run #	1	2	3	4	5	6	7	8	9	10
Seed	*56853*	*73053*	*87234*	*28867*	*15247*	*82215*	*35708*	*76259*	*59181*	*63066*
Gens req.	*844*	*1420*	*754*	*1368*	*1763*	*4424*	*1513*	*5430*	*1082*	*2958*
Achieved Fitness	*301*	*311*	*310*	*306*	*313*	*305*	*308*	*293*	*304*	*312*

Observations:

Generations required:
- Average(mean): 2155.6
- Median: 1466.5
- Standard Deviation: 1519.1

e. Discussion

Clearly our hypothesized numbers were quite off. The mean was all the way past 2000 generations and the median wasn't far behind at around 1500 generations. On top of this, although not pertaining to this experiment and not included in the datatable, the algorithm was often making an additional 9% or 10% improvement over 5000 generations. This is interesting as due to the nature of the traveling salesman problem, we assumed that after finding close to optimal paths the population would converge and mutation alone wouldn't be enough to make such a large jump in fitness. This is possibly because of two things. First our lax selection method, truncation by 50%, allowed enough diversity for organisms that had a largely different strategy. And Second, the addition of crossover may allow for large improvements to be made down the line. Our hypothesized reasoning for this is that it introduces more randomness in reproduction, allowing the model to stumble upon better solutions further along in evolution. In the future, it would be interesting to see the diversity of this population overtime to support or reject our conclusion. Additionally, possibly noting multiple advancements along the path (within 20%, within 30%, ect.) may give more insight to this algorithm and help answer some of our remaining questions.

Experiment #4

a. Description

In tournament selection, a certain number of parents will be picked and the best one will be selected to be the parent of the next generation. In this experiment, we'll examine the effect of different numbers of parents picked by tournament selection on the overall evolution.

b. Hypothesis

The more individuals are picked at one time, the higher survival pressure will be for every individual because they have to beat more competitors to reproduce. Therefore, we expect the population with higher number_parents to evolve faster towards the optimal solution.

c. Parameters

population.size=100
max.generations=1000
chromosome.length=15
mutation.rate=0.1/15
crossover.enable=true
elite.count=0
selection.method=tournament

d. Results

Notation: [S -> attrieved solution (284.38), N -> didn't arrived at solution]

2 parents are selected and compared

seed #	46582	82663	3980	39351	63116	83127	38006	46189	69682	47491
Gens req.	*288*	*157*	*235*	*469*	*217*	*106*	*280*	*83*	*302*	*339*
Highest fitness	*S*	*S*	*S*	*S*	*S*	*S*	*S*	*S*	*N*	*S*

4 parents are selected and compared

seed #	46582	82663	3980	39351	63116	83127	38006	46189	69682	47491
Gens req.	*118*	*481*	*186*	*110*	*39*	*76*	*75*	*338*	*65*	*430*
Highest fitness	*S*	*N*	*S*	*N*	*S*	*S*	*S*	*S*	*S*	*S*

6 parents are selected and compared

seed #	46582	82663	3980	39351	63116	83127	38006	46189	69682	47491
Gens req.	*135*	*105*	*177*	*546*	*95*	*88*	*681*	*69*	*53*	*528*
Highest fitness	*S*	*S*	*S*	*N*	*N*	*S*	*N*	*S*	*S*	*S*

Observations:
There is no obvious pattern on how more parents selected can improve the overall population. With the same seeded random, it's possible that the 2-parents case got the solution while the higher-number-parents case didn't (seed 39351). Even all three cases reached the solution, 4 and 6-parents didn't have advantages on the generation required to get there (seed 38006). There is also a trend that the higher the number is, the more likely for the population to not get to the final answer, though more experiments are needed to eliminate the noise.

e. Discussion
More parents selected each time doesn't seem to give any advantage to the population. Even though we thought the more parents are picked every time, the more likely better ones are selected and thus pass on their genes, it doesn't seem to be the case. From the collected data, we can only conclude that number_parent has no obvious effect on the course of evolution, but we couldn't think of an explanation because of the lack of other kinds of data that can provide more insights.

Experiment #5

a. Description

Attempt to determine the relationship between mutation rate and average number of generations taken to reach an approximate solution as well as analyzing its probability to find the ideal solution. For this experiment we will be considering being within 10% as an approximate solution and we will be comparing a few feasibly useful mutation rates: 10%, 1%, 0.1% (0.1, 0.01, 0.001). We also used a generation count of 5000 as a compromise between effectiveness and run-time. Additionally each trial had a controlled seed, so the different mutation rates had the same starting advantages or disadvantages.

b. Hypothesis

We hypothesize that the 1% will achieve fitness fastest, followed by 10%, followed by .1%. In forming this hypothesis we needed to weigh in our minds how much mutation is too much? And how much is too little? With a high enough mutation rate, the population will be unable to find a solution as fitness cannot be effectively passed down, on the other hand with too low a mutation rate progress slows down dramatically.

c. Parameters

population.size=100
max.generations=5000
chromosome.length=100
mutation.rate=MULTIPLE
crossover.enabled=true
elite.count=0
selection.method=truncation

d. Results

Plot(s):

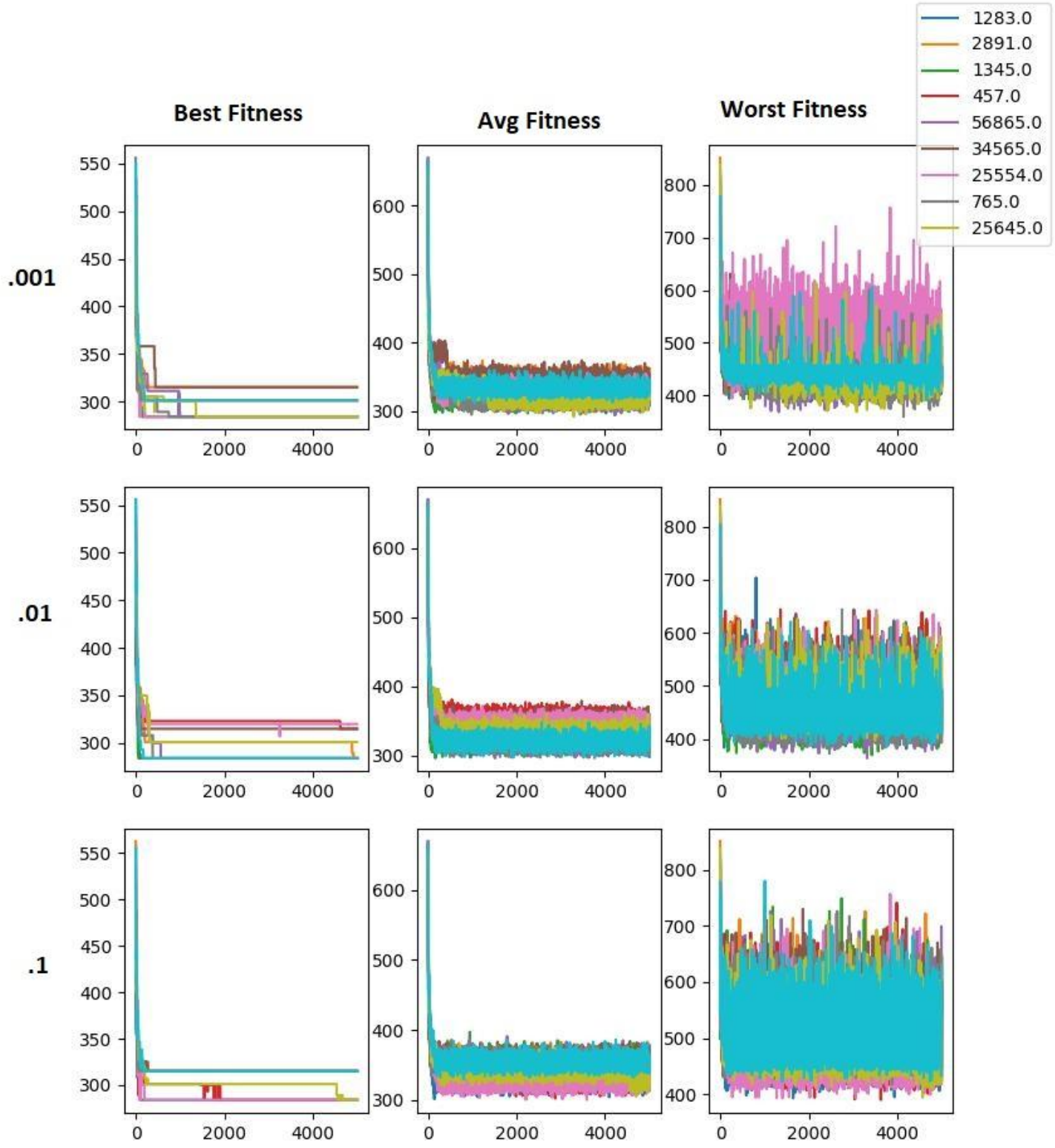


Table of Data Collected:

run #	0			1			2			3			4		
Seed	*1283*			*2891*			*1345*			*457*			*56865*		
Mutation Rate	*.1*	*.01*	*.001*	*.1*	*.01*	*.001*	*.1*	*.01*	*.001*	*.1*	*.01*	*.001*	*.1*	*.01*	*.001*
Gens req. For approx.	*43*	*56*	*81*	*N/A*	*186*	*N/A*	*N/A*	*47*	*75*	*59*	*N/A*	*87*	*N/A*	*122*	*266*
Final best fitness	*284*	*284*	*284*	*315*	*284*	*315*	*315*	*284*	*284*	*284*	*315*	*301*	*315*	*284*	*284*
Optimal Solution found?	*Yes*	*Yes*	*Yes*	*No*	*Yes*	*No*	*No*	*Yes*	*Yes*	*Yes*	*No*	*No*	*No*	*Yes*	*Yes*

run #	5			6			7			8			9		
Seed	*34565*			*25554*			*765*			*25645*			*94843*		
Mutation Rate	*.1*	*.01*	*.001*	*.1*	*.01*	*.001*	*.1*	*.01*	*.001*	*.1*	*.01*	*.001*	*.1*	*.01*	*.001*
Gens req. For approx.	*N/A*	*N/A*	*N/A*	*49*	*279*	*N/A*	*N/A*	*67*	*36*	*202*	*303*	*160*	*N/A*	*90*	*184*
Final best fitness	*315*	*315*	*315*	*284*	*306*	*323*	*315*	*284*	*284*	*284*	*301*	*284*	*315*	*284*	*301*
Optimal Solution found?	*No*	*No*	*No*	*Yes*	*No*	*No*	*No*	*Yes*	*Yes*	*Yes*	*No*	*Yes*	*No*	*Yes*	*No*

Observations:
Optimal Solution Hit Rate:
- .1: 40%
- .01: 60%
- .001: 50%

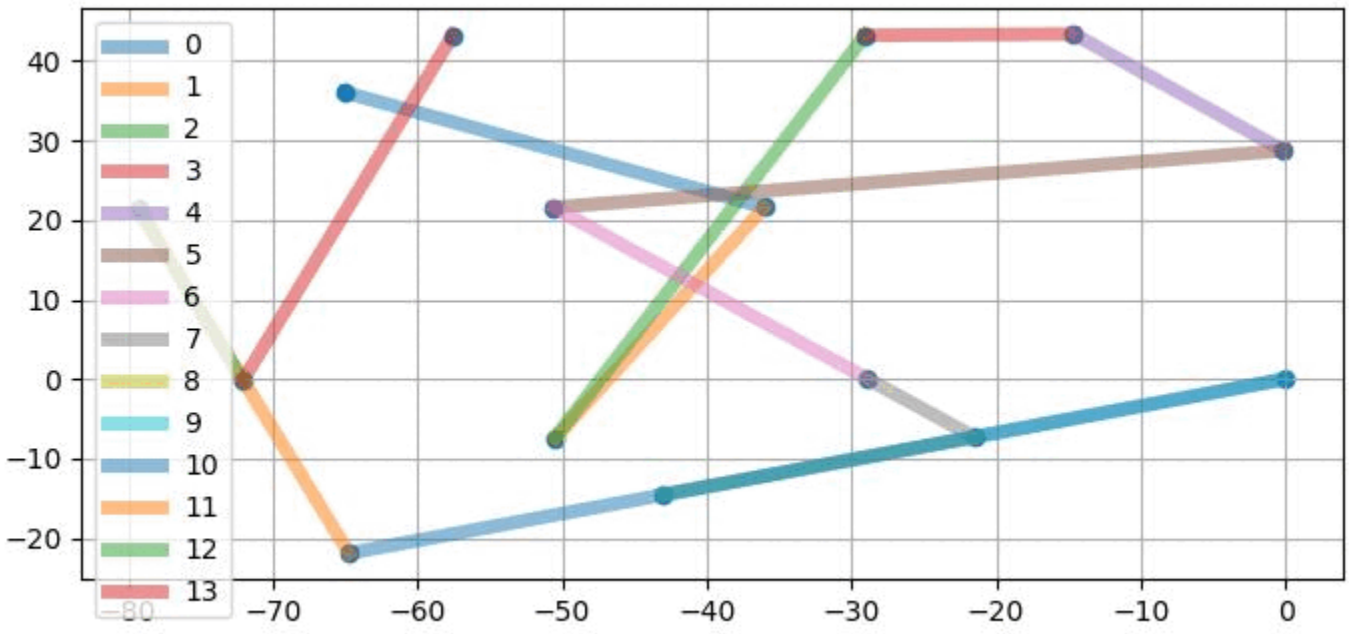
Close-To-Optimal Solution Hit Rate:
- .1: 40%
- .01: 80%
- .001: 70%

Avg Generations Required (out of runs that hit):
- .1:
  - Mean: 88.25
  - Median: 54.0
  - SD: 65.92182870643077
- .01:
  - Mean:143.75
  - Median:106.0
  - SD:94.70447455110028
- .001:
  - Mean:146
  - Median:123.5
  - SD:85.56868586112562

e. Discussion

We ended up analyzing more data than we hypothesized on, but we are good scientists so we decided not to add to the hypothesis. As far as what we did discuss before running the experiments, the data proves us wrong. While we thought it was safe to assume that the 1% mutation rate would perform best compared to its 10% and .1% competitors, this wasn't the case, or at least not the full picture. While 1% mutation did do the best in terms of hit rate for both Close-To-Optimal and the optimal solution, it was not the fastest at reaching it. While we hypothesized that the 10% mutation rate would be too high, to the point that it hinders heredity. While this may have been the case, resulting in it placing last in hit rate for both categories, we failed to realize that when it did hit, it often did it much faster than the lower mutation rates. This paints a very interesting picture that as you increase the mutation rate, you trade consistency for speed. We cant say this yet for sure, as 10 runs is still a very small sample size, but it makes sense logically and this preliminary data supports it. On the other hand, it seems that the .1% mutation rate has little to offer, placing last in both categories. This being said, something in between 1% and .1% might have merit in terms of consistency as .1% placed very close behind 1% in the consistency measurements, despite being such a low mutation rate. In the future, I would like to further explore the difference between mutation rates, possibly trying more of the values between the three we chose. Additionally, these experiments were run with crossover, as we found it really helped in finding the solution. Doing this experiment without crossover would also be very interesting, possibly then the edge mutation rates wouldn’t show as much merit.

Fun Stuff (awesomesauce??):
Once we got our algorithm working and realized it could get to the optimal solution, we got a little caught up in our amazement that we detoured for a moment and created this visualization. It shows the best path over the generations as the algorithm was able to increase its fitness. We thought it was really cool and pretty so we wanted to add it. Also we made it a gif.



Where we got the presolved traveling salesman problem from:
https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html

So much more to cover:

Summary in case you don’t have time to read extraneous material:
We had some ideas for experiments we don't have time for, but we still want to atleast make some records of. 1) Mapping Rose, 2) generating traveling salesman problems, 3) evolving the mutation rate and population size

In creating and implementing these experiments, we could not stop thinking about how much more there was to cover. We were constantly thinking of new interesting ideas that we would pursue, if we didn’t have such tight schedules. Because of this we wanted to add this section discussing the ideas we had that we thought were cool, but did not have enough time (at least right now) to pursue. The first was applying this algorithm to the buildings at Rose. This idea admittedly has the least intellectual merit, but we thought it would be a fun idea. It can be done as simply as taking a to-scale map of Rose and laying a grid over it, approximating points and plugging it into the algorithm. Conversely, if someone wanted to put in the effort, you could try to find the true distance through all the different paths and put them into a graph of Rose. This would require modification to the fitness function as well, but then you could truly find the shortest path between a set of buildings and dominate any scavenger hunt. Secondly we wanted to implement a way to generate traveling salesman problems. Then we could test our algorithm on varying complexity, and see if it can hold up. The problem is, we wouldn’t know the optimal solution. Possibly, rather than knowing the optimal solution, we could compare our solution to greedy algorithms for the traveling salesman problem or even other, more refined, dynamic algorithms. Finally, after the final experiment, the idea of evolving the mutation rate as well really intrigued me. While this would make the model much more computationally heavy, it would be interesting to find what mutation rate an evolutionary algorithm would pick. This of course begs the question, what would the mutation rate for the mutation rate algorithm be? I don't know.

We are aware that these questions have most likely already been asked, maybe even hundreds of years ago, and likely already have decades of research. We just wanted to attach them to our final report as we had a lot of fun with this algorithm and didn’t want our interest to evaporate after we move on to the next assignment.