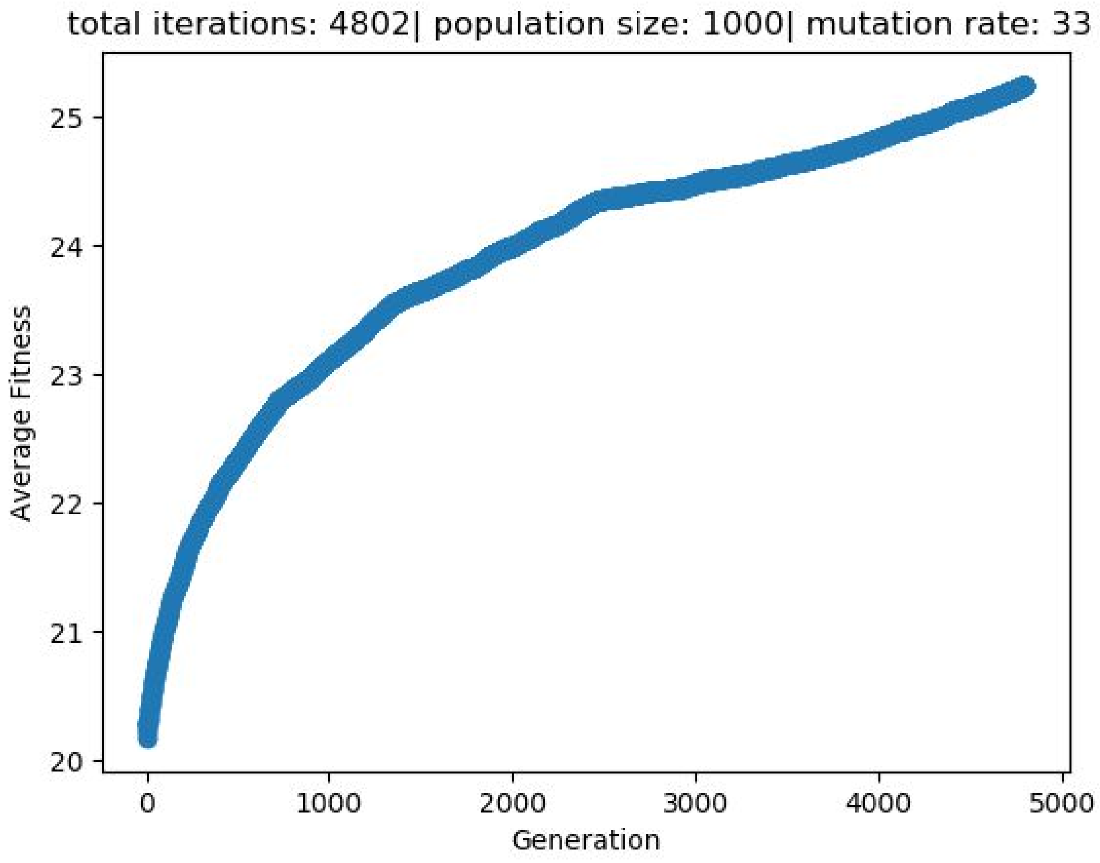
Charles Bolton
2/20/2020
Artificial Intelligence



Above is a plot, as described in the instructions, of the average fitness of my queens population per generation for a single run of the program. The graph shows a mostly logarithmic convergence to the goal fitness state, although not all of my graphs look this way. From the title you can see that the initial population was 1000 with a mutation rate of 33. I found that populations with smaller numbers did not exhibit such smooth curves if the goal state was reached quickly, but in general, any population that eventually converges exhibits this type of behavior.

Does the following: Takes user input for initial population size and mutation rate. Then it does the following for up to 10,000 iterations:

1) calculates the fitness score for each individual.
2) calculates the fitness probability for each individual.
3) calculates the bins for each individual based on the fitness proportion selection algorithm.
4) selects two random individuals with probability based on the above.

5) breeds these two members, generating two new children.
6) sorts the population based on fitness and culls the least two fit members of the population.
7) if one of the children is the goal state, the program prints out the queens on the board and halts, also printing out a plot of the graph.

With my technique I discovered that larger populations actually take longer on average (more time and more iterations) than smaller populations. My eugenic culling technique leads to fast convergences, especially with small populations from 10-100 members. The culling also keeps the initial population constant at all times. Some of my peers reported very slow or non-terminating programs by generating an entirely new population from each previous population. I think this doesn't make sense with respect to the metaphor of genetics, since we want to preserve the genes that are the most fit; rather than replacing the entire population anew each time, my algorithm keeps the fittest from all generations. This is similar to the way in which dominant alleles will persevere through many generations, while weaker genetic traits will die or disappear more quickly. This possibility of "immortality" is not opposed to genetics, it is merely opposed to anthropomorphising an already contrived metaphor for reaching a goal state. Since this technique rarely doesn't converge to a goal state with populations in the range of 10-1000, I think it is a rather successful application of a genetic algorithm.

In addition to the final puzzle configuration and graph, my program will output a text file called 'out.txt' with each run, which shows, for each generation, the most fit individual. Below is an example of explicit members of the population from initial population to goal state:

```
fittest member of population at iteration 0:
(0.000942133187881192, [4, 6, 6, 5, 8, 4, 3, 2]). Score: 0.0
fittest member of population at iteration 100:
(0.001189117199391172, [4, 7, 4, 8, 1, 6, 2, 5]). Score: 25
fittest member of population at iteration 200:
 (0.0011630611770179111, [8, 4, 3, 6, 2, 5, 1, 4]). Score: 25
fittest member of population at iteration 300:
(0.0011434321258690084, [2, 7, 5, 8, 8, 5, 1, 6]). Score: 25
fittest member of population at iteration 400:
(0.0011745042236978814, [5, 5, 2, 4, 6, 8, 3, 7]). Score: 26
fittest member of population at iteration 500:
(0.0011641964805444857, [3, 8, 8, 5, 2, 4, 1, 7]). Score: 26
fittest member of population at iteration 600:
(0.0011528399769432005, [2, 2, 6, 8, 1, 8, 4, 7]). Score: 26
fittest member of population at iteration 700:
(0.0011431586352444601, [2, 2, 6, 8, 1, 8, 4, 7]). Score: 26
```

```
fittest member of population at iteration 800:
(0.001136512654631289, [4, 2, 4, 8, 7, 5, 3, 6]). Score: 26
fittest member of population at iteration 900:
 (0.0011313693921065227, [4, 2, 4, 8, 7, 5, 3, 6]). Score: 26
fittest member of population at iteration 1000:
(0.0011248107289638762, [1, 6, 8, 5, 2, 4, 6, 7]). Score: 26
fittest member of population at iteration 1100:
(0.001162290142057684, [3, 6, 8, 4, 1, 7, 5, 2]). Score: 27
fittest member of population at iteration 1200:
 (0.001156118866147127, [3, 6, 8, 4, 1, 7, 5, 2]). Score: 27
fittest member of population at iteration 1300:
 (0.0011495231607629428, [3, 6, 8, 4, 1, 7, 5, 2]). Score: 27
fittest member of population at iteration 1400:
(0.0011446012972148034, [3, 6, 8, 4, 1, 7, 5, 2]). Score: 27
fittest member of population at iteration 1500:
(0.0011417938850594156, [3, 6, 8, 4, 1, 7, 5, 2]). Score: 27
fittest member of population at iteration 1600:
 (0.0011382799325463744, [3, 6, 8, 4, 1, 7, 5, 2]). Score: 27
fittest member of population at iteration 1700:
(0.0011356944561285438, [3, 6, 8, 4, 1, 7, 5, 2]). Score: 27
fittest member of population at iteration 1800:
(0.001132360342224459, [3, 6, 2, 5, 7, 1, 4, 8]). Score: 27
fittest member of population at iteration 1900:
 (0.0011281965569112486, [3, 6, 8, 4, 1, 7, 5, 2]). Score: 27
fittest member of population at iteration 2000:
 (0.0011249531269530437, [2, 6, 1, 6, 8, 3, 7, 4]). Score: 27
fittest member of population at iteration 2100:
 (0.0011215885016408423, [5, 1, 6, 4, 2, 7, 3, 8]). Score: 27
fittest member of population at iteration 2200:
 (0.0011182902584493041, [2, 6, 1, 3, 5, 8, 4, 1]). Score: 27
fittest member of population at iteration 2300:
(0.0011150574048071364, [2, 6, 1, 3, 5, 8, 4, 1]). Score: 27
fittest member of population at iteration 2400:
 (0.0011106997408367272, [2, 6, 1, 3, 5, 8, 4, 1]). Score: 27
fittest member of population at iteration 2500:
(0.0011084198858738042, [2, 6, 1, 3, 5, 8, 4, 1]). Score: 27
fittest member of population at iteration 2600:
(0.0011073288766763729, [2, 6, 1, 3, 5, 8, 4, 1]). Score: 27
fittest member of population at iteration 2700:
(0.0011062853396705727, [2, 6, 1, 3, 5, 8, 4, 1]). Score: 27
fittest member of population at iteration 2800:
(0.0011050628248680063, [2, 6, 3, 7, 2, 8, 5, 1]). Score: 27
fittest member of population at iteration 2900:
(0.0011043848167539268, [2, 6, 3, 7, 2, 8, 5, 1]). Score: 27
fittest member of population at iteration 3000:
```

```
(0.0011026709139916686, [2, 6, 8, 3, 1, 4, 7, 1]). Score: 27
fittest member of population at iteration 3100:
(0.0011011868346996207, [2, 6, 8, 3, 1, 4, 7, 1]). Score: 27
fittest member of population at iteration 3200:
 (0.0011001548366066335, [2, 6, 8, 3, 1, 4, 7, 1]). Score: 27
fittest member of population at iteration 3300:
(0.0010991695163654128, [2, 6, 8, 3, 1, 4, 7, 1]). Score: 27
fittest member of population at iteration 3400:
(0.001097159575764964, [2, 6, 8, 3, 1, 4, 7, 1]). Score: 27
fittest member of population at iteration 3500:
(0.001095690284879474, [2, 6, 3, 7, 4, 1, 3, 5]). Score: 27
fittest member of population at iteration 3600:
(0.0010945798029756355, [2, 6, 3, 7, 4, 1, 5, 2]). Score: 27
fittest member of population at iteration 3700:
(0.0010931174089068825, [2, 6, 3, 7, 4, 1, 5, 2]). Score: 27
fittest member of population at iteration 3800:
(0.00109143827310211, [2, 5, 7, 1, 3, 8, 6, 8]). Score: 27
fittest member of population at iteration 3900:
(0.0010893246187363835, [6, 2, 5, 1, 4, 4, 8, 3]). Score: 27
fittest member of population at iteration 4000:
(0.001087219135056777, [4, 7, 5, 8, 3, 1, 6, 2]). Score: 27
fittest member of population at iteration 4100:
(0.0010846858428410734, [2, 6, 3, 7, 4, 1, 5, 5]). Score: 27
fittest member of population at iteration 4200:
(0.0010827284757589126, [2, 6, 3, 7, 4, 1, 5, 5]). Score: 27
fittest member of population at iteration 4300:
 (0.001080691642651297, [3, 6, 8, 4, 1, 7, 5, 2]). Score: 27
fittest member of population at iteration 4400:
 (0.0010780164497324921, [5, 8, 4, 7, 3, 8, 6, 1]). Score: 27
fittest member of population at iteration 4500:
(0.0010758686643289768, [3, 8, 2, 5, 7, 1, 4, 6]). Score: 27
fittest member of population at iteration 4600:
(0.0010741565881604075, [4, 7, 1, 5, 2, 8, 6, 3]). Score: 27
fittest member of population at iteration 4700:
(0.0010714710901226238, [5, 1, 6, 4, 2, 7, 3, 6]). Score: 27
fittest member of population at iteration 4800:
(0.0010691375623663579, [5, 5, 2, 4, 6, 8, 3, 1]). Score: 27

Goal State has been found! After: 4802 iterations.

This is the configuration of queens found [4, 8, 1, 3, 6, 2, 7, 5]
```

**Sample terminal output from another run:**

```
Please enter initial population size in the range (10, 1000): 100
Please enter mutation rate as a percentage in the range (0, 100): 33

Begin up to 10000 iterations: Breeding...
i:..0
i:..100
i:..200
i:..300
i:..400
i:..500
i:..600
i:..700
i:..800
i:..900
i:..1000
i:..1100

Goal State has been found! After: 1116 iterations.

This is the configuration of queens found [4, 8, 1, 5, 7, 2, 6, 3]


___ ♕ ___ ___ ___ ___ ___ ___

___ ___ ___ ___ ♕ ___ ___ ___

___ ___ ___ ___ ___ ___ ♕ ___

___ ___ ___ ♕ ___ ___ ___ ___

♕ ___ ___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___ ___ ♕

___ ___ ___ ___ ___ ♕ ___ ___

___ ___ ♕ ___ ___ ___ ___ ___
```