TECHNIQUE

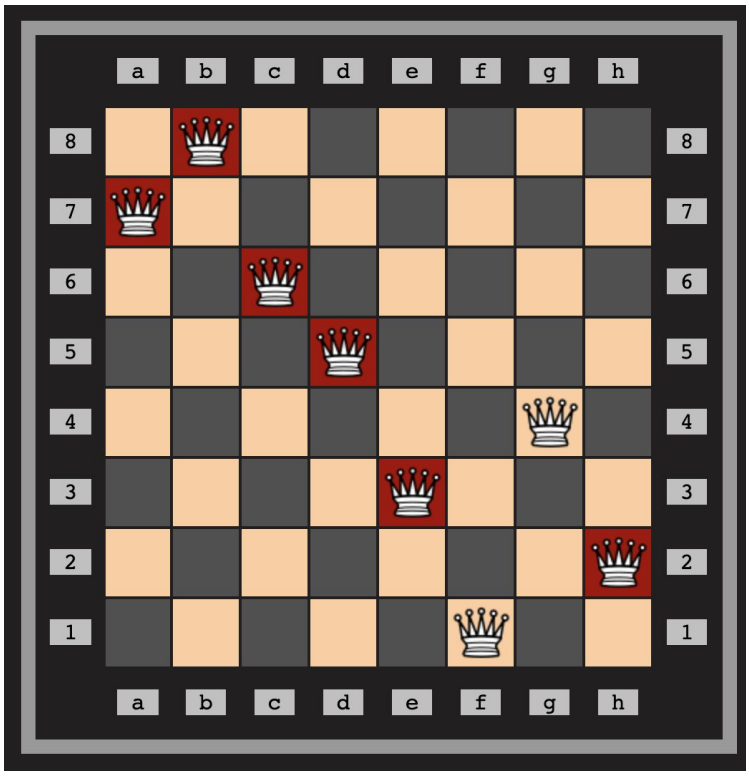
The Representation

The representation featured one list of size 8. Each 'index' in that list represented a column, and the element inside represented which row was inside. The element inside the row is considered to be unique, and many of the functions inside the implementation treated it as such, as will be explained in the **Reproduction** section below.

For example, if a list was represented as

[6, 7, 5, 4, 2, 0, 3, 1]

It would be expressed in an 8x8 board as



These constraints on the representation cleared up many factors.

1. There may be no more than 8 queens in one representation.
    a. By having a 'fixed' size, it helps to ensure that ALL queens will be included in the representation
2. There may only be one queen per column
    a. The idea of having the fixed size additionally allows us to assume each queen will have its own column. With this constraint, there's no need to check for multiple queens in a column, as it is not possible with the representation.
3. The unique style only allows for one queen per 'element'.

a. The implementation of the reproduction allows for each list to give each queen a unique row. This was actually sort of accidental, but it figured out to prove extremely efficient as can be seen in the results below.

The Fitness Function

In my implementation, the fitness function calculated the amount of attacking pairs. This can occur in 3 possible ways:

1. Queens attack via the same column. (Vertically)
2. Queens attack via the same row. (Horizontally)
3. Queens attack via a diagonal axis. (Diagonally)

Considering the representation mentioned above, there is ***NO*** possible way for the queen to ever attack in the same column, since each space in the list represents a column, and the size of the list is fixed.
Additionally, with the 'accidental' unique-ness to each element, there is no possible way for any two queens to share a row. Therefore, there is no possible collisions horizontally.

That leaves a diagonal attack for the pairs of queens. Given the representation, it is known which column and which row each queen is located in. Therefore, the graph below shows how simple it is to calculate which queens conflict in diagonals. By calculating the sum of the column and row, or the difference of the column and the row, it is able to calculate which diagonal the queens lie on, and consider if any of the other queens will attack another one. This can be visually proven in the diagram to the right.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	8
2	2	3	4	5	6	7	8	9
3	3	4	5	6	7	8	9	10
4	4	5	6	7	8	9	10	11
5	5	6	7	8	9	10	11	12
6	6	7	8	9	10	11	12	13
7	7	8	9	10	11	12	13	14

Direction:
//

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	1	2	3	4	5	6	7
1	-1	0	1	2	3	4	5	6
2	-2	-1	0	1	2	3	4	5
3	-3	-2	-1	0	1	2	3	4
4	-4	-3	-2	-1	0	1	2	3
5	-5	-4	-3	-2	-1	0	1	2
6	-6	-5	-4	-3	-2	-1	0	1
7	-7	-6	-5	-4	-3	-2	-1	0

Direction:
\\

The Reproduction

Initial Population:
The initial population is any shuffled arrangement of all integers from [0-7], per the representation format.

Crossover:
The idea behind my implementation of crossover is similar to an example of modern socio-economic marriage. The rich marry the rich, the poor marry the poor. Except instead of checkbooks, we're talking about fitness. So, the 'most fit' of the generation would mate with the second-most fit of the generation. The second-most fit of the generation would mate with the third-most fit. This would trickle down until the very last case, in which case is the princess-peasant marriage. The most unfit of the generation mates with the most fit of the generation, in expectation that the next generation's offspring will be more fit.

However, cross-over would occur through a two-point split. What would happen is that Parent A would select a random section of its genes to keep. From there, it would iterate through Parent B's genes, removing any duplicates of those genes. It would then adapt the residual of Parent B's genes, inserting Parent A's genes where appropriate. This would guarantee that the representation stays fixed, in terms of size, which is the necessary constraint. However, this implementation was also adapted so that it could contain unique elements for each row, providing a benefit in speed.

Mutation:
Mutation was guaranteed for all, after all it's evolution. Every single member of every single generation would find that they had two elements in their columns swapped at random.

FINDINGS

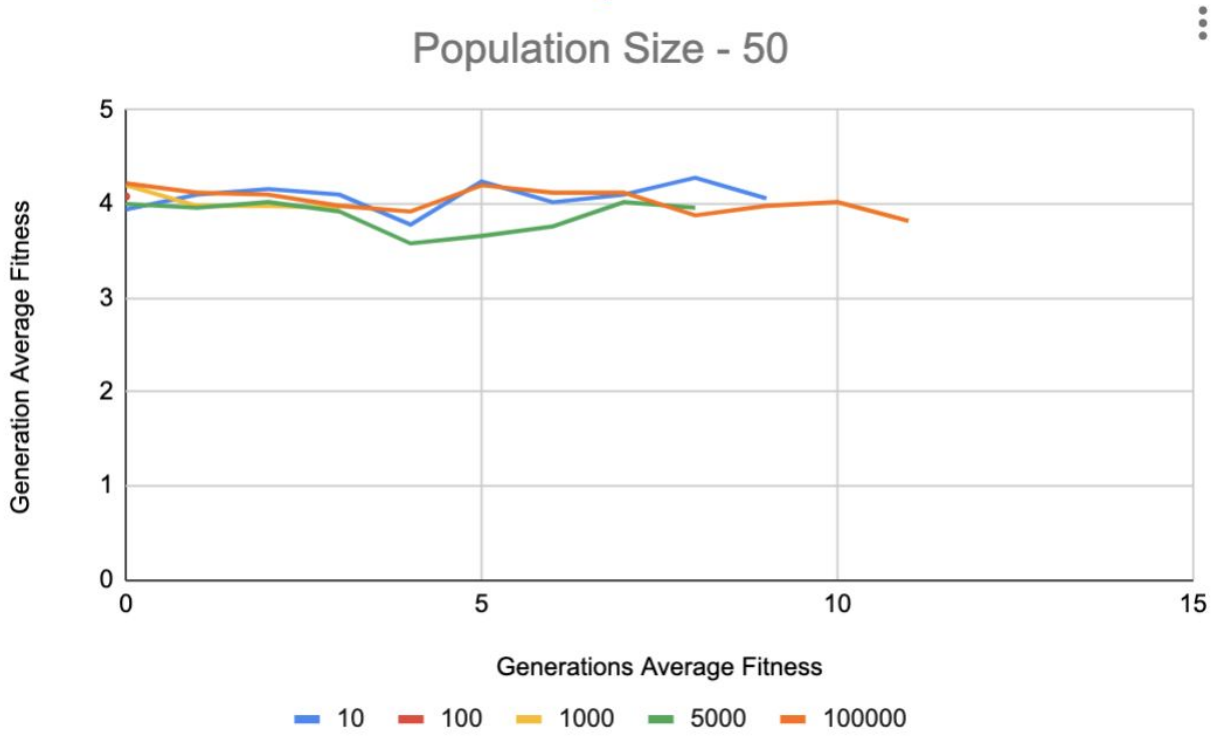
Findings
Initially, when I first programmed this, I found that there were INCREDIBLY slow results before I had the unique 'row' trait implemented. I figured I had already written over half of it, so I would just continue, and I found incredible gains by this.

What then occurred is depicted in the following graphs. These show the generational average fitness versus the generation it is in, with the colors each representing the maximum amount of iterations in between each. Each graph shows how big the initial population was, as labeled in the graph title.

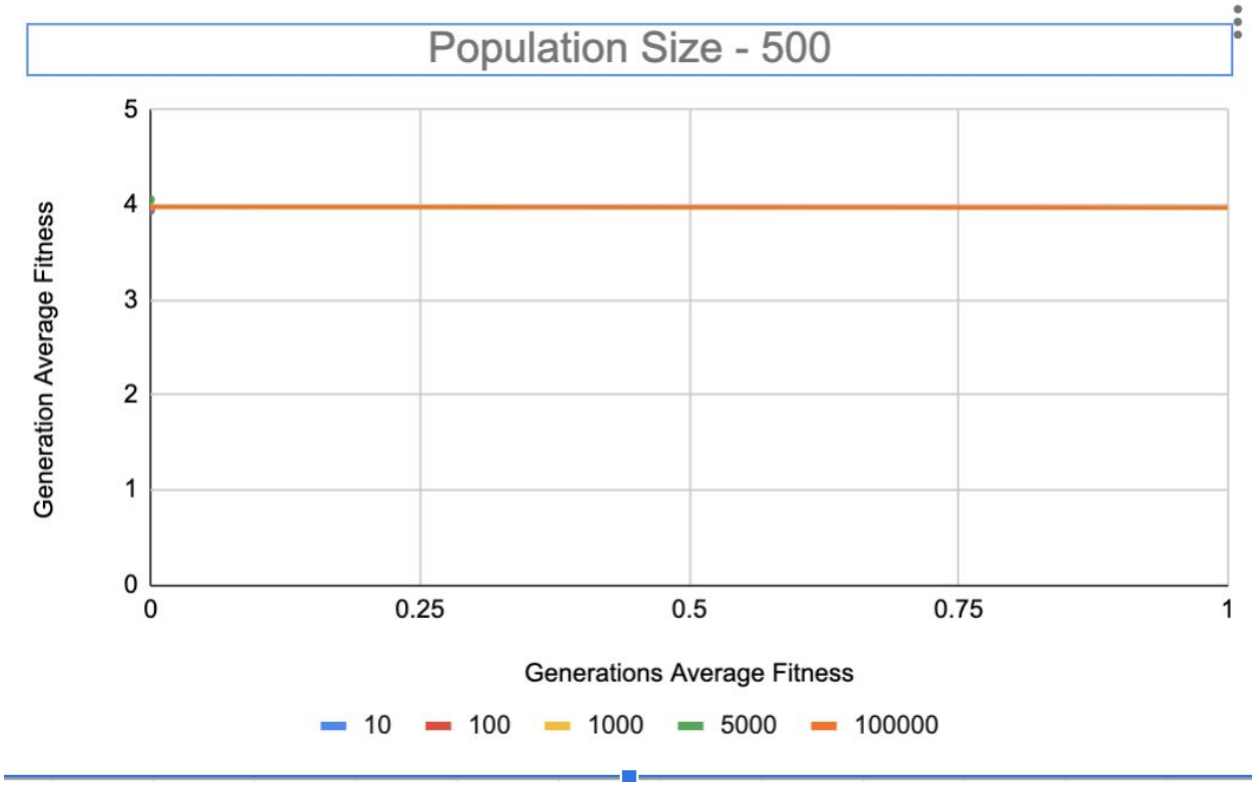
The following parameters were used for the Genetic Algorithm:
Varying Population Sizes of: 5, 10, 50, 100, 500, 1000, 5000.
Variyng Maximum iterations of: 10, 100, 1000, 5000, 10000


Population Size - 5
Generation Average Fitness
0
2
4
6
0
20
40
60
Generations Average Fitness
10
100
1000
5000
100000


Population Size - 10
Generation Average Fitness
0
2
4
6
0
20
40
60
Generations Average Fitness
10
100
1000
5000
100000


Population Size - 50
Generation Average Fitness
0
1
2
3
4
5
0
5
10
15
Generations Average Fitness
10
100
1000
5000
100000


Population Size - 50
Generation Average Fitness
0
1
2
3
4
5
0
10
20
30
Generations Average Fitness
10
100
1000
5000
100000



** For populations of size 1000 and 5000, they actually resulted to be instantaneous.*

CONCLUSIONS

As can be seen in the graph above, there was no ‘unsuccessful’ termination. Each genetic algorithm was able to reach its own steps, despite the population size. While I think that this may be unrealistic, and perhaps my results were lucky, I do think that there was some sort of success thanks to the representation. I believe that by having fixed, unique columns and rows, there is already two of the three possible ‘clashes’ fixed.

I also found that by increasing the initial population size, it increased computational time significantly, but it also found results quicker. It would be very safe to assume that there is some sort of trade off, maybe depending on the problem, but it would be useful to increase the population size to something computable. I found that when there was 100 in the population, all but one trial ran under 10 generations.

Lastly, I think that the selection of parents could definitely impact the ‘average function fitness’. While I used the elites stay with elites, misfits stay with misfits, I definitely think a more ‘random’ or at least ‘elite-focused’ approach may actually give better results. The problem is that the average fitness fluctuates, with no promising downward trends. I’d be curious to explore how changing crossover may in the future lead to some more guaranteed downward trends and faster results.