8 Queens Genetic Algorithm Report

Name: David Shaulov

ID: 317005403

Chromosome representation:

- I chose to represent the position of the queens as a list of length 8, where $list[i]$ is the row of the queen on the $j'th$ column.
- For example, for $[6, 3, 7, 4, 1, 8, 2, 5]$, the queen on the third column is on row 7.
- $8x8$ matrix representation is inefficient since if two queens occupy the same column it is an invalid solution.

Fitness function implementation:

- The fitness function measures the amount of pairs of queens that threaten one another – queens that are on the same row or diagonal.
- The less queen that threaten one another the higher the fitness.

Selection type:

- I have implemented both roulette and elitism selection.

Crossover types:

- I implemented single-point, two-point and uniform crossover types.

Mutation implementation:

- Given mutation rate p, with probability p the mutation functions swaps the places of two queens.
- Picking a random index and changing it to some random number is inefficient since it may choose a number already present in the list – meaning chooses a row that already has a queen on it, which is an invalid solution.

Experiment method:

- Since there is an element of probability, I ran each experiment 10 times and took the average of results.
- I disregarded solutions generated by chance from initial population generation.

Low mutation, low population experiments:

Population Size	Number of Generations	Crossover Type	Mutation Rate	Average Unique Solutions	Average Time (Seconds)
50	1,000	Single-point	0.05	0	0.27
50	1,000	Single-point	0.2	0	0.27
50	1,000	Uniform	0.05	0	0.29
50	1,000	Uniform	0.2	0	0.29
50	10,000	Single-point	0.05	0	2.7

50	10,000	Single-point	0.2	0	2.9
50	10,000	Uniform	0.05	0	2.9
50	10,000	Uniform	0.2	0	2.9
100	1,000	Single-point	0.05	1	0.6
100	1,000	Single-point	0.2	1	0.7
100	1,000	Uniform	0.05	1	0.6
100	1,000	Uniform	0.2	1	0.7
100	10,000	Single-point	0.05	1	6.6
100	10,000	Single-point	0.2	1	6.8
100	10,000	Uniform	0.05	1	6.8
100	10,000	Uniform	0.2	1	7.0

<u>Low mutation, high population experiments:</u>

Population Size	Number of Generations	Crossover Type	Mutation Rate	Average Unique Solutions	Average Time (Seconds)
150	1,000	Single-point	0.05	2	1.15
150	1,000	Single-point	0.2	1	1.21
150	1,000	Uniform	0.05	2	1.20
150	1,000	Uniform	0.2	2	1.17
150	10,000	Single-point	0.05	3	11.21
150	10,000	Single-point	0.2	2	11.12
150	10,000	Uniform	0.05	2	11.58
150	10,000	Uniform	0.2	2	11.92
300	1,000	Single-point	0.05	5	3.42
300	1,000	Single-point	0.2	3	3.66
300	1,000	Uniform	0.05	4	3.22
300	1,000	Uniform	0.2	4	3.37
300	10,000	Single-point	0.05	7	31.81
300	10,000	Single-point	0.2	14	34.01
300	10,000	Uniform	0.05	2	35.30
300	10,000	Uniform	0.2	16	43.80

<u>High mutation, high population experiments:</u>

Population Size	Number of Generations	Crossover Type	Mutation Rate	Average Unique Solutions	Average Time
300	10,000	Single-point	0.3	14	36.41
300	10,000	Single-point	0.4	19	33.78
300	10,000	Uniform	0.3	16	33.13
300	10,000	Uniform	0.4	22	33.63

Brute force experiments:

- For the brute force method, I simply generated random lists of the elements [1,2,3,4,5,6,7,8] a certain number of times and counted how many unique solutions were created:

Repetitions	Average Unique Solutions Found	Average Time
1,000	0	0.005 seconds
10,000	0	0.05 seconds
100,000	1	0.5 seconds
1,000,000	6	5 seconds
10,000,000	36	50 seconds

Conclusions:

- Best result for the genetic algorithm is 22 solutions after 33.63 seconds.
- Best result for the brute force algorithm is 36 solutions after 50 seconds.
- If a “smarter” brute force were used – where permutations are generated instead of random lists – it would find all 92 solutions in 50 seconds – outperforming the genetic algorithm.