CSC 180 – Project 4

Solving N-Queens Using Genetic Algorithms

By: Quinn Roemer & Logan Hollmer

Model/Code Design

```python
#Helper function, returns True if the tuple or reverse tuple exists in the dictionary
def checkPair(first, second, conflictDict):
  conflict = (first, second)
  rConflict = (second, first)

  #Check to see if they exist or not
  if conflict in conflictDict:
    return True
  elif rConflict in conflictDict:
    return True
  else:
    return False
```

A Helper Function Quinn Wrote for his Evaluation Function

```python
def evaFitness(individual):
  conflicts = 0
  for i in range(len(individual)):
    for j in range(i+1, len(individual)):
      if not i==j:
        if individual[i]==individual[j]:
          conflicts+=1
        elif (individual[i]==abs(i-j)+individual[j] or individual[i]==-abs(i-j)+individual[j]):
          conflicts+=1

  # Return the number of conflicts
  return (conflicts,)
```

Logan's Evaluation Function

Evaluation Functions

- Quinn's evaluation function focused on efficiency and could perform an evaluation without examining every unique pair of Queens
- Logan chose the more obvious route and simply compared each Queen with each other. Since the number of possible Queen's is small, this works just fine.

Model/Code Design

```
# Constants
NUM_GEN = gen
NUM_SELECT = 5000
NUM_CHILDREN = 10000
CXPB = 0.1
MUTPB = 0.9

# Evolve the population using eaMuPlusLambda()
pop, log = algorithms.eaMuPlusLambda(pop, toolbox, NUM_SELECT, NUM_CHILDREN, CXPB, MUTPB, NUM_GEN, stats=stats, halloffame=hof, verbose=True)
```

Logan's training code for *eaMuPlusLambda()*

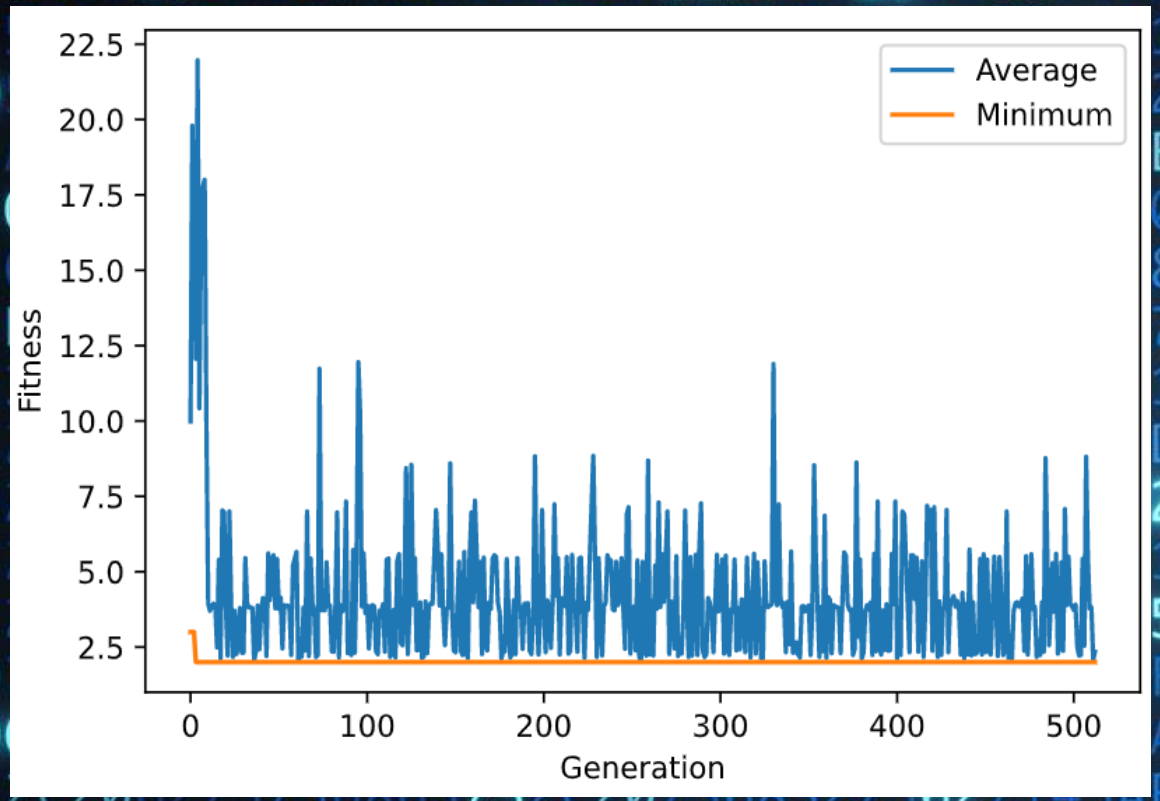
```
N = 32
pop = toolbox.population(n=100)
hof = tools.HallOfFame(maxsize=1)
generationsTrained = 0
log = tools.Logbook()

#Train for a generation
while True:
  pop, newLog = algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=1, stats=stats, halloffame=hof, verbose=False)
  log.record(gen=generationsTrained, avg=newLog.select("avg"), min=newLog.select("min"))
  generationsTrained +=1
  print(f'Generation: {generationsTrained}, best found: {hof[0].fitness}')

  if hof[0].fitness.values[0] == 0:
    print(f'Solution found after {generationsTrained} generations.')
    break
```

Quinn's training code for any N using *eaSimple()*

Population Training

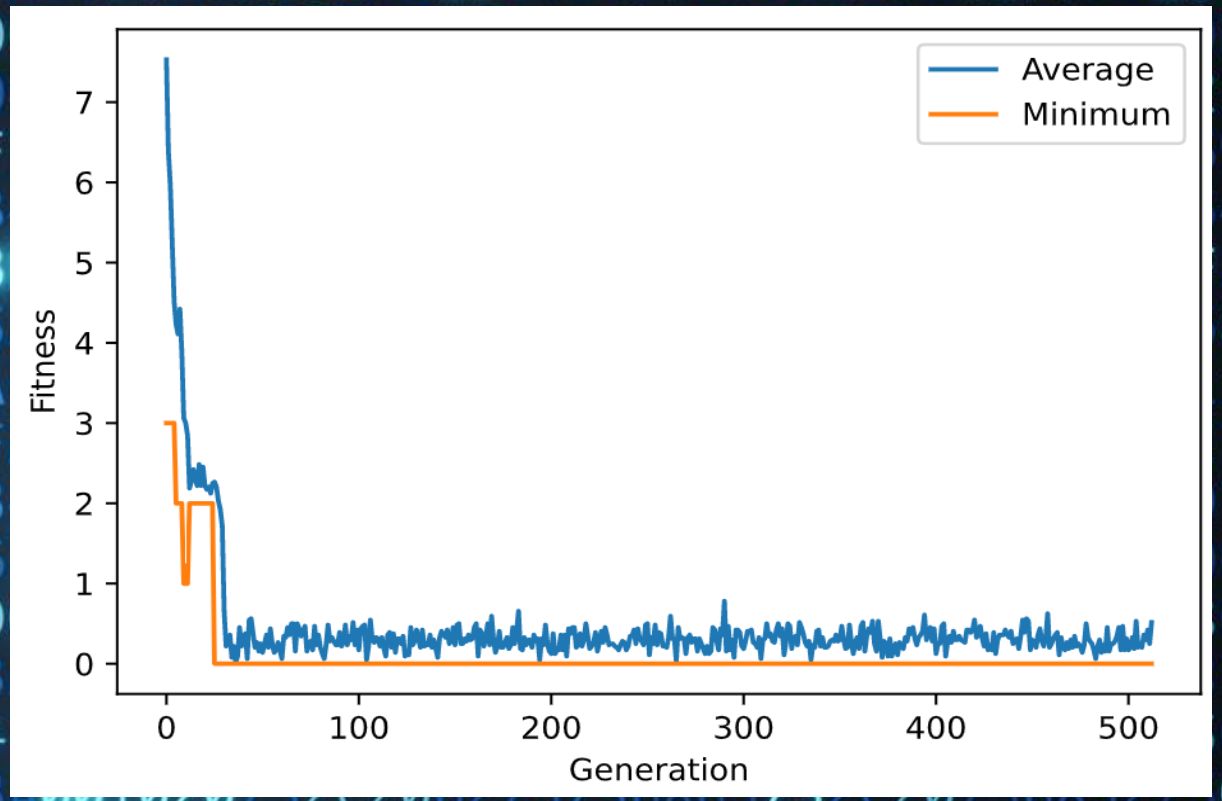
- Quinn stuck to the *eaSimple()* algorithm.
- Logan experimented with other algorithms such as the *eaMuPlusLambda()* algorithm.

Findings/Results



Position-Index-Based Representation

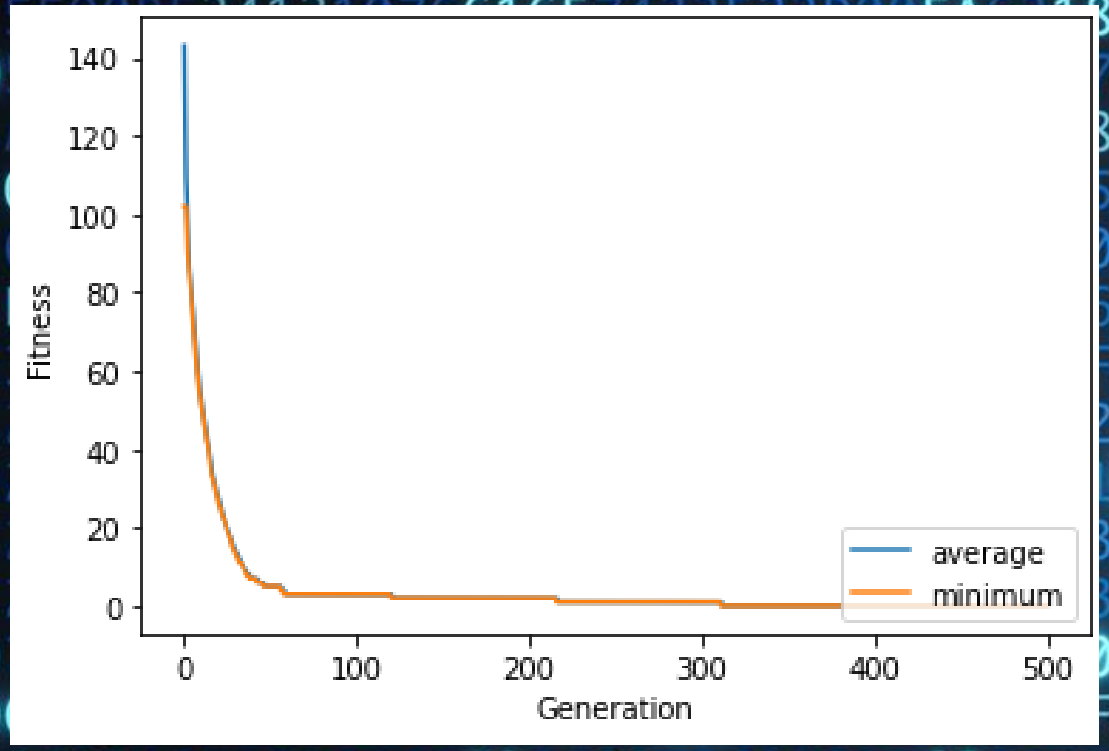
- Population of 64, trained for 512 generations
- Resulting Fitness: 2
- Best Solution Found: [23, 1, 6, 50, 60, 35, 45, 16]



Row-Index-Based Representation

- Population of 64, trained for 512 generations
- Resulting Fitness: 0
- Best Solution Found: [5, 2, 6, 3, 0, 7, 1, 4]

Findings/Results



124x124 Queens Solution:

Logan's successful solution to the 124x124 Queens's problem involved several changes to the strategies used to evolve the populations, the changes are listed below:

- Mutation chance increased to more quickly introduce new genes into the population.
- Selection tournament size increased so that only the best individuals were chosen to create the next generation (this resulted in the min and average being almost equal).
- *eaMuPlusLambda()* algorithm used to evolve the population.
- Population size of 10k, trained for 500 generations.

Solution:

[4, 86, 59, 65, 46, 84, 7, 110, 44, 58, 17, 69, 83, 106, 40, 122, 80, 34, 109, 113, 54, 103, 57, 55, 74, 52, 79, 88, 94, 35, 117, 120, 50, 32, 78, 24, 19, 0, 76, 9, 45, 61, 2, 123, 112, 26, 42, 63, 93, 97, 81, 118, 20, 30, 6, 98, 119, 82, 11, 16, 111, 121, 92, 36, 49, 1, 28, 107, 90, 115, 48, 5, 105, 53, 18, 31, 91, 116, 60, 102, 72, 89, 101, 95, 77, 12, 62, 75, 33, 22, 100, 38, 87, 104, 13, 21, 8, 3, 37, 10, 23, 41, 66, 70, 29, 114, 96, 56, 51, 67, 47, 39, 43, 85, 14, 99, 71, 27, 25, 15, 68, 108, 64, 73]

Task Division, Challenges Encountered, and what we Learned

Task Division

- Both group members fully finished the project, accomplishing all necessary tasks alone. Each group member also took a stab at solving the general N-Queens problem.
- The report was equally split with Quinn writing the first two sections, and Logan writing the last two sections.
- Quinn scripted and created the presentation

Challenges Encountered

- Solving for boards greater than 32x32 proved difficult without changing certain strategies.
- The computation time for larger boards with larger populations was several hours.

What we Learned

- In general, the Row-Index-Based representation created better results, since no board could be generated where multiple Queens existed on the same row (a fundamental factor of all solutions).
- A higher population count generally produced better results as a more diverse set of genes helped the population find a solution faster.
- Creating a representation that guaranteed no Queens would be placed on the same row or column would’ve allowed solutions to be found quicker.