Gabriel Sampang
CS441
2/20/2022

Program 2 Write up

Instructions on how to compile and run my code:
My code for Programming Assignment #2 is all in 1 file, main.cpp. I have programmed in C++. I have used headers that make my code nonportable. For this reason, the only way I am sure it will run is if compiled on the linux servers.

This compiles on linux using g++.
(ex: promt> g++ main.cpp -o program2).

Then it's as simple as running the executable generated by the compiler.
(ex: promt> ./program2).

My program takes user input to specify what the population size and number of iterations will be for a given run. As it is currently coded, the population size must be even.
In order to speed up the process of running it, I have included sample input files. These can automatically be fed into the program with '<'.
(ex: prompt> ./program2 < input1.txt)

I found the most success with the input found in inputLong.txt.

My findings, techniques, and conclusions:
When I plotted the average fitness over generations, I found that the average fitness didn't follow a clear pattern. In some cases like in the one plotted below, the overall average would decrease. In other cases, it would simply stay roughly the same.
I also found that my program could not consistently find solutions.

I believe these findings to be the result of the way that I chose the parents for breeding the next generation. The assignment document states two ways that the parents can be chosen: randomly or proportionally to their fitness. I chose the latter.
In my design, I wanted to guarantee that the 2 most fit parents of a given generation would reproduce, and then the 2 most fit and so on. To implement this, I had the breeding done in pairs in order of fitness. What I didn't account for is that this method allows for the least fit individuals to also reproduce to the next generation.

If I had more time I would implement this program in a way that removes the least fit from the next generation. This should gradually raise the average fitness with each successive generation. In fact, after some experimental work with it I found this very trend. However there is not

Gabriel Sampang
CS441
2/20/2022

enough time to completely verify and validate the experimental code so the program attached is the original polished version.

<u>Parameters:</u>
To create the initial population, I divided the total population into 8. Each section would be made up of permutations of a randomly generated array with the starting value being unique to that section.

For example: a population of 8 may look like:

1. 12345678
2. 23456781
3. 34567812
4. 45678123
5. 56781234
6. 67812345
7. 78123456
8. 81234567

This is done in order to get good variation in the population. Note, that duplicate individuals may occur if a given section is larger than the number of permutations

For example, a section 8 may look like: (duplicated individuals because of no more permutations)

1. 11111111
2. 11111111

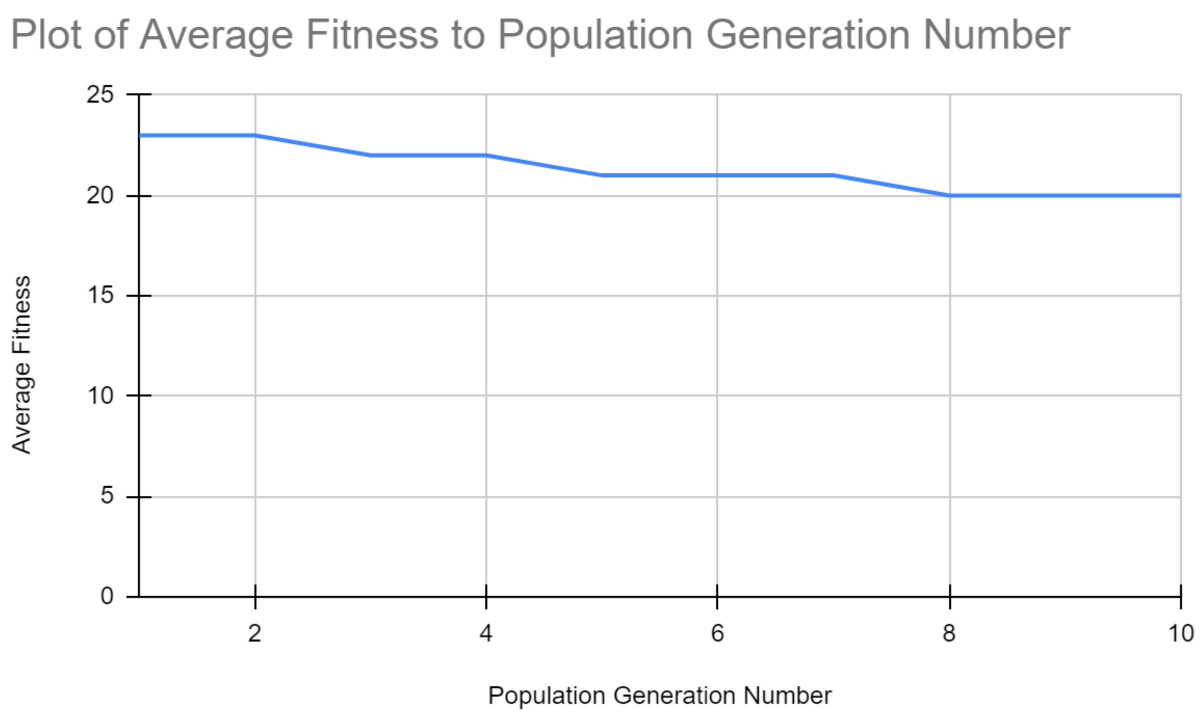
<u>Fitness function:</u>
The fitness of a given state is defined as the number of pairs of queens not attacking each other. The maximum fitness which indicates a solution is 28. This is because there are 8 choose 2 pairs of queens in the 8-Queens problem, which is 28 pairs.

For simplicity, the fitness function was calculated by going through each of the 28 pairs and checking if they are in the same column, row, or diagonal. If they are not in any of these, then the queens are not attacking each other. In practice, there may be another piece in between a given pair that blocks the pair from attacking each other but since in the goal state this cannot be, this case is not considered when calculating the fitness of a state.

Gabriel Sampang
CS441
2/20/2022

<u>Mutation:</u>
The mutation of a gene was determined by a variable MutationPct. There is to be a mutation every MutationPct generations.
(i.e. if MutationPct = 1, then there would be a mutation every generation)
I used a MutationPct of 20 which means mutations every 20 generations, meaning there would be a 5% chance of a given individual having a mutation.
The gene was randomly selected using a random number generator.

Plot) Average fitness to population generation number:



The above plot is for a run that had success finding solutions over 10 generations

Example individuals across generations:

Below are individuals from the run used in the above plot.

Example 1: one of the solutions found
State: 6 1 5 2 8 3 7 4
Fitness Value: 28

Gabriel Sampang
CS441
2/20/2022

Example 2: one of the solutions found
State: 7 2 6 3 1 4 8 5
Fitness Value: 28

Below are individuals across generations from a simpler run. Similar results can be generated from running the app with input from input3.txt.

```
Initial generation: --------------------------------------------------

Member: 1
State: 5        8       3       6       1       7       2       4
Fitness Value: 23

Member: 2
State: 4        7       2       5       8       6       1       3
Fitness Value: 25

Printing generation 2: --------------------------------------------------

Member: 1
State: 4        8       3       6       1       7       2       4
Fitness Value: 24

Member: 2
State: 5        7       2       5       8       6       1       3
Fitness Value: 26

Printing generation 3: --------------------------------------------------

Member: 1
State: 5        7       2       6       1       7       2       4
Fitness Value: 23

Member: 2
State: 4        8       3       5       8       6       1       3
Fitness Value: 24

Printing generation 4: --------------------------------------------------

Member: 1
State: 4        8       3       5       8       7       2       4
Fitness Value: 22
```

Gabriel Sampang
CS441
2/20/2022

```
Member: 2
State: 5        7       2       6       1       6       1       3
Fitness Value: 25

Printing generation 5: ---------------------------------------------

Member: 1
State: 5        7       2       6       1       6       1       4
Fitness Value: 23

Member: 2
State: 4        8       3       5       8       7       2       3
Fitness Value: 21


Simulation results:
No solutions found.

Statistics -----------------------------------------------------
Generation: 1          Average Fitness: 24
Generation: 2          Average Fitness: 25
Generation: 3          Average Fitness: 23
Generation: 4          Average Fitness: 23
Generation: 5          Average Fitness: 22
```