**Report about GA when applied over N queen and TS problems**

**by Ephi Frankel and Yarden**

**Part1 – N Queen Problem**

Introduction

A very well known problem in the domain of optimization problems is the N queen problem. Given a board at size N X N, find a placement for N chess queens such that no queen will threat another queen.

In this report we aim to show the results when trying to solve this problem for N=8 with GA. We will also use a randomized CS (Constraints Satisfaction) algorithm that finds an optimal solution as our baseline results comparison.

Baseline approach

There are many possible ways to find a viable solution for the N queen problem. Let us review some of them in terms of speed/performance in order to select the most reasonable as baseline comparison to the GA algorithm.

1. Naïve Brute Force: Generate all possible permutations of N queens at the N X N board for each permutation, check if there exists a threat in the board, select a permutation that has no threats.

Complexity analysis: Factorial at N

Algorithm is complete – that is, a solution is promised to be found.

1. Recursive Algorithm: Recursively generate boards that maintain the valid solution throughout their construction (if there is a violation, try another path).

Complexity analysis: Exponential at N

Algorithm is complete – that is, a solution is promised to be found.

1. Randomized Algorithm: Generate a random permutation that is a possible solution for the N queen problem such that there is a queen at each row. For each queen from row i: N-2 to 0, move it on the same row if it threats any queen at row i+1,i+2, .., N-1, if a valid position is found then check for the queen at the upper row, if not, re-apply the algorithm, if valid position were found for all rows return the solution.

Complexity analysis: C\*N^2 where C is the number of random permutations needed to find a solution. Note: C is not bounded – this implies that the algorithm is not complete, the algorithm will only halt if a solution was found, otherwise, will continue indefinitely. However, if the algorithm halts, the solution is optimal. Also, in practice C is usually small.

Note: We might bound C with a factorial of N by only generating new permutations at each iteration of the algorithm, in this case the algorithm will be complete, with worst case complexity of Factorial at N. However, in this report we are using the unbounded version, and thus, it is an incomplete algorithm.

Since in practice C is small, then, the runtime of (3) is preferred over the other options. So, we chose it as our baseline.

Baseline results:

When the baseline approach is applied, optimized solution might be found as fast as 0.004 seconds. However, due to the reasons discussed above, it is possible that no solution might be found at all.

Fig. 1 shows an outcome given the baseline method.

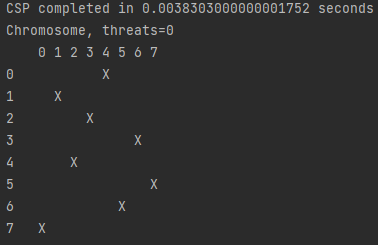


Fig. 1 – An example for a baseline result.

GA approach

As discussed throughout the lectures, GA requires the following main building blocks in order to be applied:

*Chromosome:* a possible solution of the problem.

*Fitness function:* measures how “fit” is a chromosome.

*Selection mechanism:* defines how does 2 chromosomes are selected for cross-over.

*Cross-over function:* defines a re-combination over 2 chromosomes into 2 offspring.

*Mutation function*: applied over a chromosome causes at a very low probability random change at the chromosome.

Additionally, *elitism* is a mechanism in which a percent of the strongest chromosomes passed into the next generation at the expanse of the same percent from the weakest chromosomes.

8-Queen GA

*Chromosome:*

First, we consider the following 1-D representation for the problem such that each queen is allocated a single row. To be precise, we say that each queen belongs specifically to row and that queens by definition cannot share rows. By this definition we reduce a dimension of the problem since now no 2 queens can ever be located on the same row, so for each we define the location of at row . So, we have that as our representation.

Next, we consider that then each can be represented with exactly 3 bits (000 = 0, 001 = 1, …, 111 = 7). Due to this, we define the Chromosome to be such that is the 3 bit representation for the location of , is the 3 bit representation for the location of , …, is the 3 bit representation for the location of . We note that by construction the chromosome is valid for any sequence of bits , then, **there exists no cross-over/mutation that could cause such chromosome to become invalid.**

*Fitness function:*

For the fitness function we consider the number of threats generated on the board. We do it as follows:

For each count the number of that are either on the same columns or on the same diagonals such that

We say that and are on the same column if =

We say that and are on the same diagonal if = or =

By this definition **a chromosome is more fit if the number returned by this function is lower.**

*Selection mechanism:*

The selection mechanism we used is a roulette wheel selection. Before applying this selection, we are inverting the fitness function such that each chromomere fitness F will correspond to where is a number close to 0 to avoid division by 0.

*Cross-over function:*

The cross-over function we chose is the uniform binary cross-over. That is, given two chromosomes and the cross-over function creates two offsprings:

and such that at probability p=0.5, and with probability 1-p, for all i.

*Mutation function:*

The mutation function we chose is binary flip, that is, given chromosome , each is flipped with probability p.

*Elitism:*

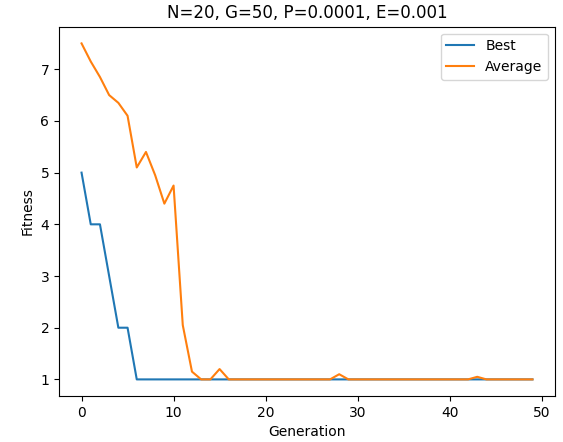
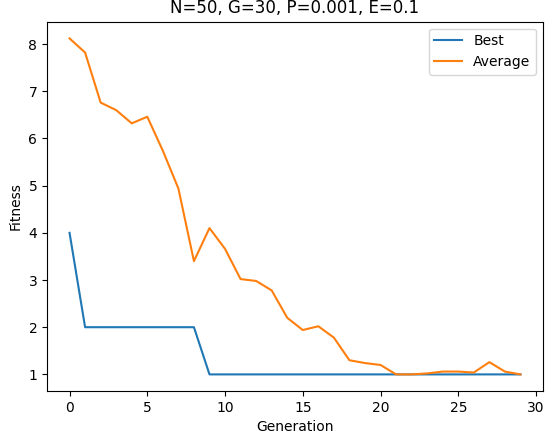
Elitism is defined over a percentage of the population, but at least 2 are always selected.

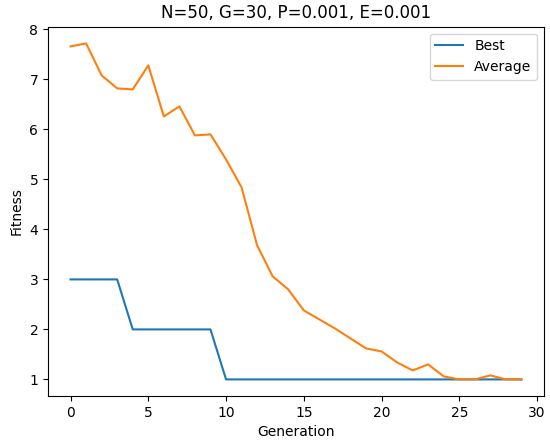
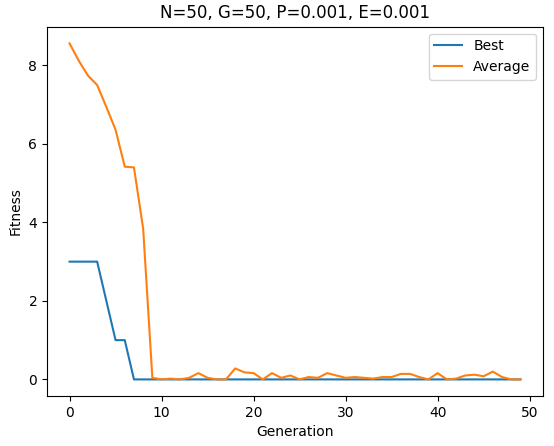
Experiments

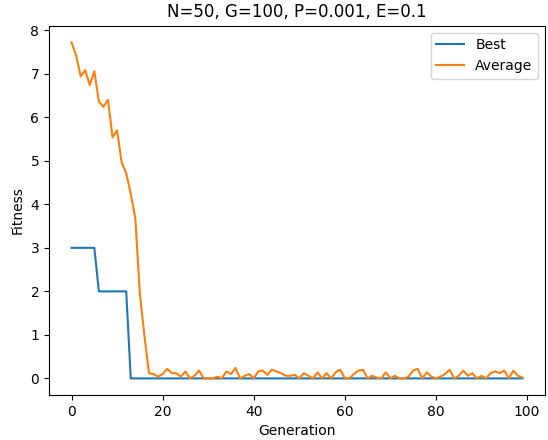
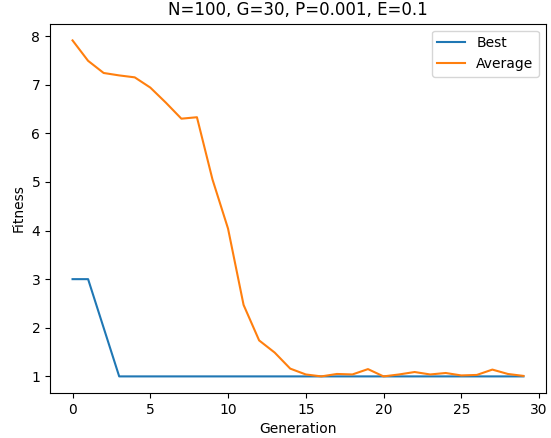
The GA paradigm provides many hyper-parameters to fine tune when performing experiments. In fact, we may consider the chromosome, fitness, cross-over, mutation and selection mechanism as hyper-parameters as well. We are justifying the chromosome design and fitness we selected by their simplicity and correctness, so we have not experimented with different possibilities. Additionally, we justify the selection of cross-over and mutation function due to the knowledge we gained in the lectures, that is, that the standardized cross-over and mutation of bitwise chromosome is always valid is as described above.

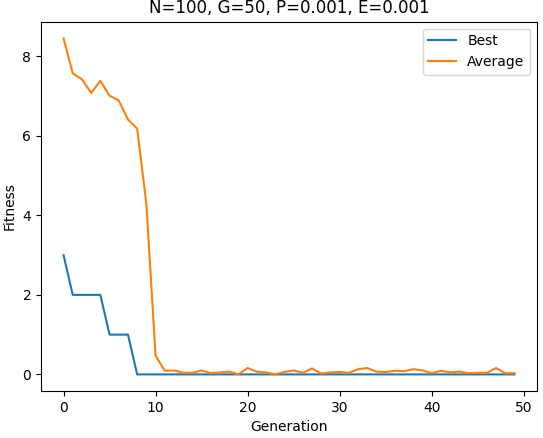
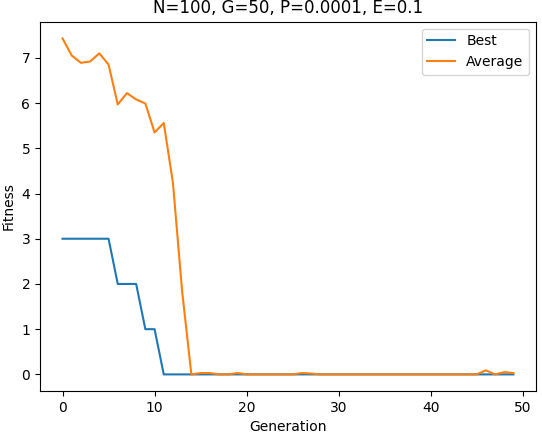
The hyper-parameters we adjusted then are as follows: N = the size of the chromosome population, G = the number of generations, P = the probability for mutation, E = the elitism percentage. Below are the promising results when we experimented over following:

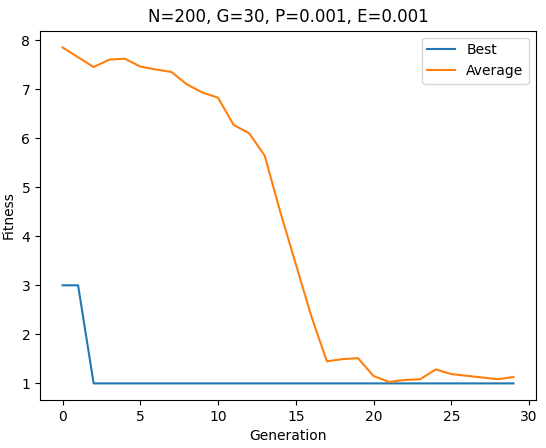
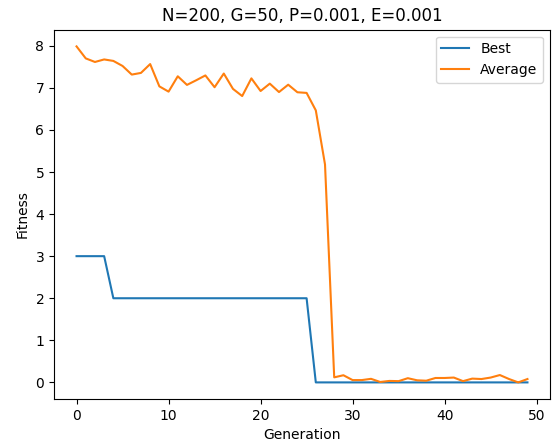
N = [10, 20, 50, 100, 200, 500]  
G = [10, 30, 50, 100, 200]  
P = [0.01, 0.001, 0.0001]  
E = [0.1, 0.001]

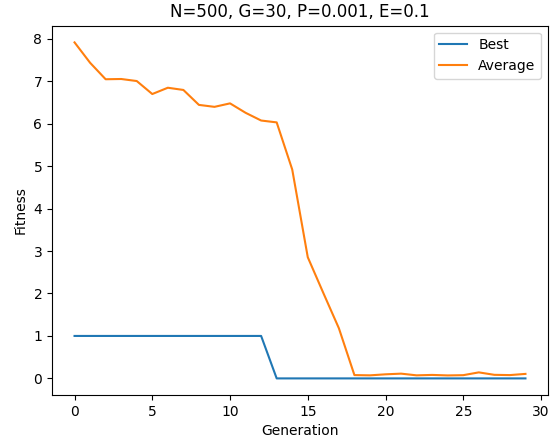
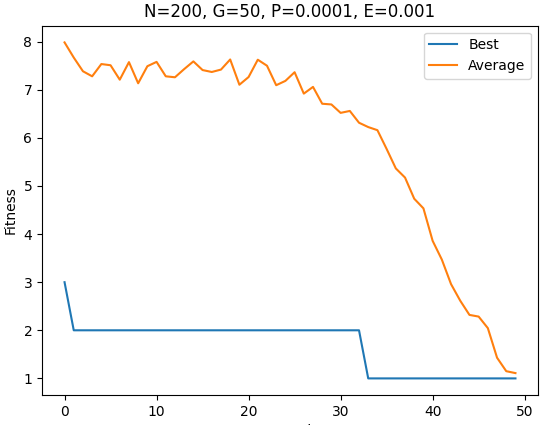
 

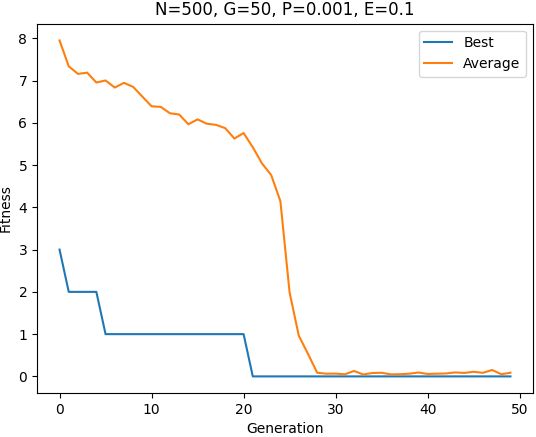
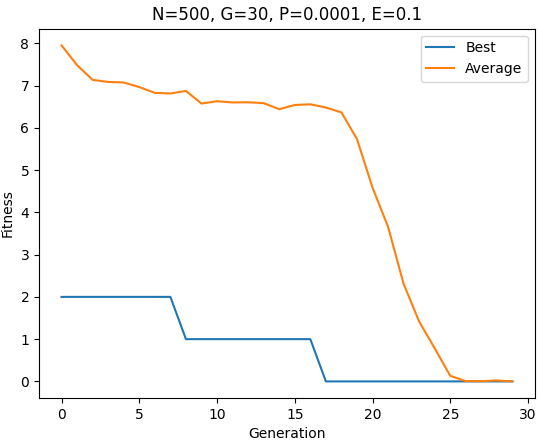
 

**** ****

****

****

Results analysis

As can be seen over the graphs of the promising results, many of the configurations reached sub-optimal solutions (i.e: fitness = 1). In many of the results where an optimal solution was reached, it was probably due to a powerful mutation that almost immediately spread throughout the population (for example, consider the result for N=200, G=50, P=0.001, E=0.001). It seems that as the population size increases, so does the stability of the algorithm resulting with gradual improvement (for example, consider the result for N=500, G=30, P=0.0001, E=0.1). However, in terms of speed, even the quickest configurations (e.g: N=20, G = 50) took 0.02 seconds which is more than an order of magnitude slower than the average run time of baseline algorithm, while the slower and more stable configurations at the GA perform in the order of dozens of seconds.

Conclusion

By analyzing the results, we may understand that the baseline algorithm is on average better than using the GA approach for the 8-Queen problem. However, we consider the following in mind: first, that the baseline approach is not complete, that is, it MAY stuck (or at least timeout with an invalid result). Second, the GA algorithm MOST of the time will return a good approximation of the optimal result (i.e: solution with only a single threat). We also note that the GA was applied here for N=8, under the assumption that as N increases the probability of generating boards such that a viable solution is reachable from them decreases, we have that the C in our baseline algorithm complexity analysis increases. Then, due to the increase of C we assume that for big enough N the performance of the GA in both terms of speed and approximation will be even better than that of the baseline algorithm presented in this report.

**Part2 – TS Problem**