PROGRAMMING ASSIGNMENT

Student Report
Artificial Intelligence
CS F407
CSIS

Submitted By:
Arshika Lalan
2017B3A70620G

The 8 Queens Problem

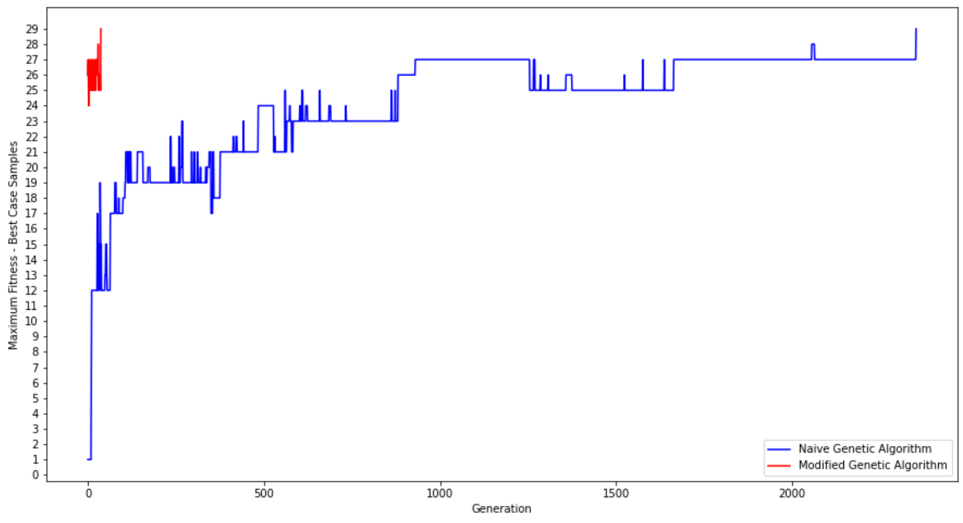
Naïve Genetic Algorithm Implementation: The algorithm had a population size of 20, with all 20 individuals of the initial population set to [1,1,1,1,1,1,1,1]. The mutation rate was kept small, at 1%, so that for the population size of 20, the expected individuals that underwent mutation was 0.2. A textbook crossover function and mutation function implementation was done. It was observed that the naïve genetic algorithm did perform decently well most of the times, and found a solution all the time in approximately 4000 generations. It could get stuck in local minimas however, driving up the generations to as high as 50,000. The naïve algorithm was run 10 times, and the fitness values were sampled and averaged.

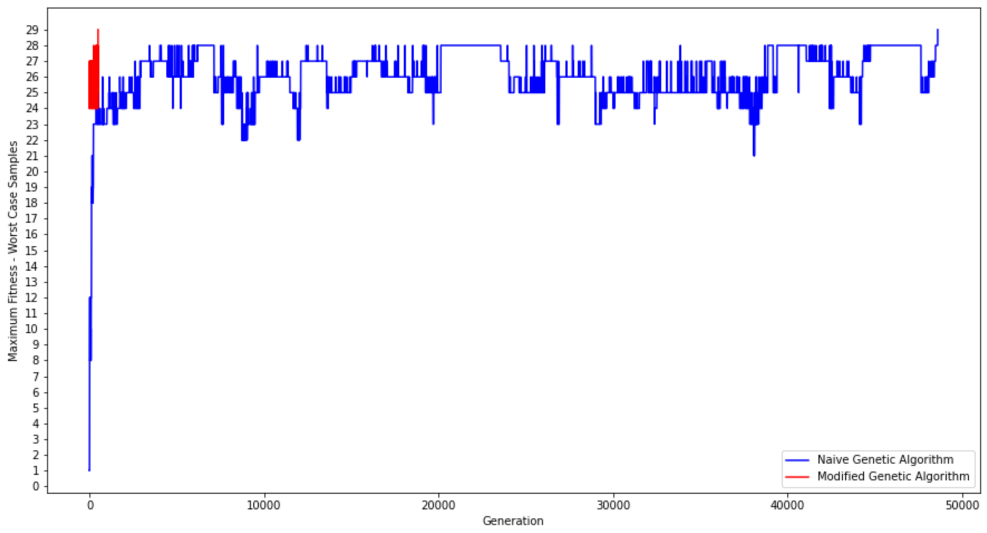
Modified Genetic Algorithm Implementation: A major drawback of the population structure proposed in the question was the lack of diversity in the initial population. This led to slow convergence towards the solution for the first few generations till the population became diverse enough. Another important thing to note is that the nature of the 8 queens problem is such that even changing one queen position in a good solution could lead several queens into attacking position and bring down the fitness value. Therefore, the naïve mutation function of changing just one queen position isn't necessarily the best choice. To tackle both of these problems, I have introduced a more aggressive mutation function, wherein every mutation a completely random child is generated which carries forward no genetic material from either of its parent. This introduces some much needed diversity early on, and prevents the degradation of the fitness value due to the previous mutation function.

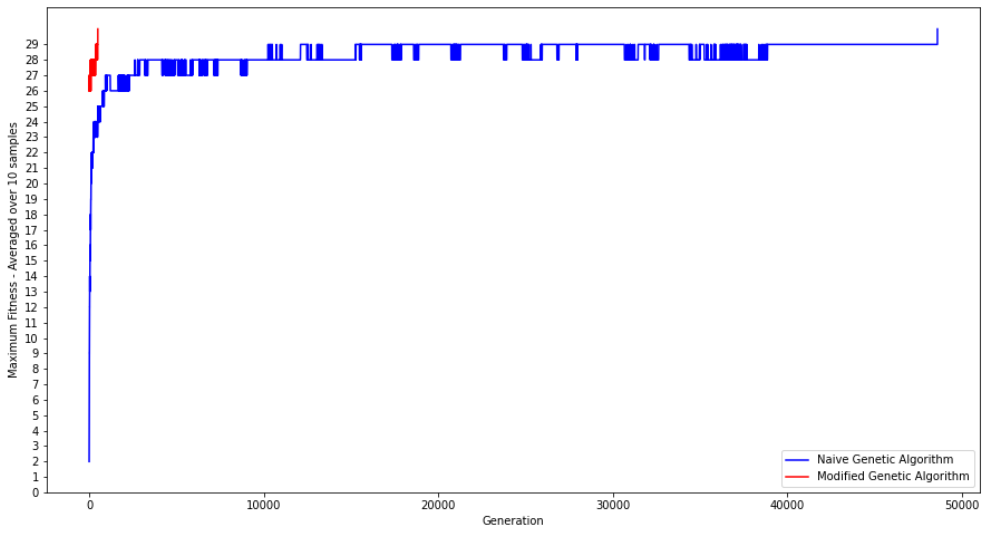
As explained before, since small changes in an individual could lead to huge differences in results which needn't necessarily be good, I have also used a different crossover function. The function compares the two parents, and finds the common elements (common positions on the board). It passes on these common elements to the child as it is, and generates the rest of the positions randomly. This leads to rapid convergence in the algorithm, as the children retain the genetic information which forms part of the perfect solution. Though this crossover function discourages diversity, the more aggressive mutation function makes up for it and ensures that the algorithm doesn’t get stuck in local minimas like the naïve genetic algorithm. Another modification was made to the generation of the next populations, where the weakest half of the population was killed and replaced with stronger half, thus keeping the population size consistent. This survival of the fittest technique helped to further increase convergence to the optimal solution. Finally, to make full use of the diversity, the population size was increased to 300, and mutation rate to 35%. The final algorithm converged extremely rapidly as expected, and reached a solution in as low as 20-30 generations. On average, the algorithm took around 60 generations to converge to a solution. Moreover, the algorithm rarely got stuck in local minimas. Over several runs of the algorithm, it got stuck in a local minima only once and still managed to get an optimal solution in around 1000 generations. 10 fitness values were sampled and averaged for the modified genetic algorithm.

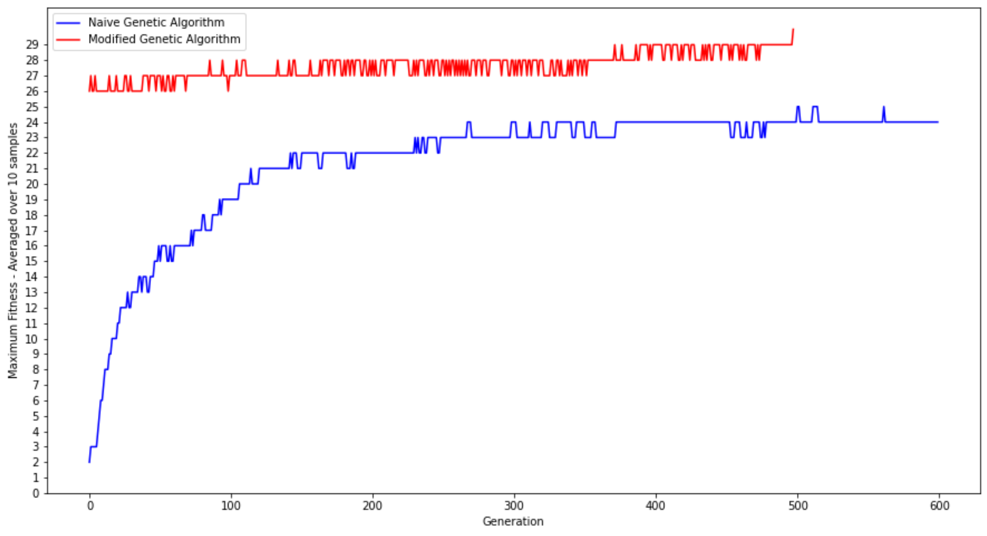
Comparison Graphs

The below graphs show that the modified algorithm severely outstrips the naïve algorithm in terms of performance. 10 fitness values were sampled for both the algorithms, and the best case/worst case performances are compared. The 10 fitness values were then averaged and compared, and it is seen that the modified algorithm performs better in all 3 cases – best case, worst case, as well as average case. For ease of viewing, a zoomed-in graph of the average case is also attached at the end.




Maximum Fitness - Worst Case Samples
Generation
Naive Genetic Algorithm
Modified Genetic Algorithm


Maximum Fitness - Averaged over 10 samples
Generation
Naive Genetic Algorithm
Modified Genetic Algorithm


Naive Genetic Algorithm
Modified Genetic Algorithm
Maximum Fitness - Averaged over 10 samples
Generation

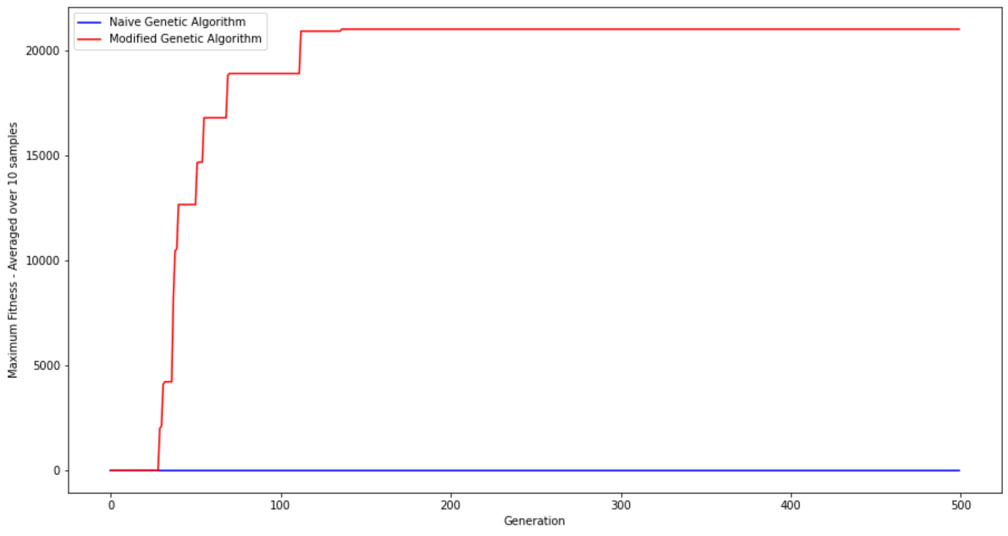
The Travelling Salesman Problem

Naïve Genetic Algorithm Implementation: The algorithm had a population size of 20, with all 20 individuals of the initial population set to [0,1,2,3,4,5,6,7,8,9,10,11,12,13]. The mutation rate was kept small, at 1%, so that for the population size of 20, the expected individuals that underwent mutation was 0.2. The fitness value was taken as (10000/pathDistance+1). The crossover function generated offsprings by choosing a random subset from parent1, and then inserting the cities from parent2 which were not observed in the subset in the remaining positions in order (As described in the question). The mutation function swapped any two cities randomly. The naïve genetic algorithm performed extremely poorly for the travelling salesman problem, and did not converge to a path without infinities even after running for 70,000 generations. The lack of diversity in the population is the most likely cause of this, which led to unfit population being produced every generation. The naïve algorithm was run 10 times, and the fitness values were sampled and averaged.

Modified Genetic Algorithm Implementation: The low population size and low mutation rate led to slow computations, and hindered convergence. Therefore, the population size and the mutation rate both were increased. Moreover, the mutation function was changed from swapping two random cities to swapping two adjacent cities, which led to a drastic improvement in performance. Another modification introduced was that of elitism, wherein the best n of the population were carried forward as it is to the next generation. This was done because, unlike the 8 queens problem, small changes in good solutions can nudge us towards an optimal solution in the Travelling Salesman Problem. These measures were successful, and while the naïve algorithm couldn’t reach even a valid path in 70,000 generations, the modified algorithm was able to reach optimal solutions within 200-500 generations. I had also made modifications to the crossover function, one of them being the famous cyclic crossover technique. However, these did not optimize the algorithm further, and hence I retained the crossover function from the naïve algorithm due to its simplicity.

Comparison Graphs

The below graph shows that the modified algorithm severely outstrips the naïve algorithm in terms of performance. 10 fitness values were sampled for both the algorithms, and the average case performances are compared. Since there wasn’t much variation in the fitness values over generations, comparison for the best\worst case is skipped for the economy of space.


Naive Genetic Algorithm
Modified Genetic Algorithm
Maximum Fitness - Averaged over 10 samples
Generation