Assignment 1 Artificial Intelligence Group 7

Write Up

By,
Ganesh Prasanna Balakrishnan
Nishant Shah
Rishi Khajuriwala
Mohamad El-Rifai

Part 1. Heavy N-queens Problem

1) How large of a puzzle can your program typically solve within 10 seconds using A*? Using greedy hill climbing?

Greedy Hill-Climbing:

For Greedy hill climbing, 31 Queens is the maximum number for which the program solves the puzzle in 10 seconds. We got the solved puzzle in 5.49 seconds. But this is not the case every time, as it gets stuck at local minima. Our program mostly (9 out of 10 times) gives solution for 20-25 queens within 10 seconds but for an assured solution within ten seconds we would recommend solving a 15-20 Queens problem.

```
rishi@rishi-khajuriwala:~/Downloads/AI/assignment1$ python3 nqueens.py
Enter the number of queens you want to play with:31
Restart 1
Solved
Solved
The solved state is:
{0: 17, 1: 7, 2: 22, 3: 13, 4: 1, 5: 21, 6: 27, 7: 29, 8: 23, 9: 5, 10: 0, 11: 12, 12: 10, 13: 16, 14: 26, 15: 24, 16: 2, 17: 4
, 18: 20, 19: 11, 20: 25, 21: 14, 22: 3, 23: 18, 24: 28, 25: 9, 26: 6, 27: 15, 28: 19, 29: 8, 30: 30}
Time elapsed 5.493551445000776
rishi@rishi-khajuriwala:~/Downloads/AI/assignment1$
```

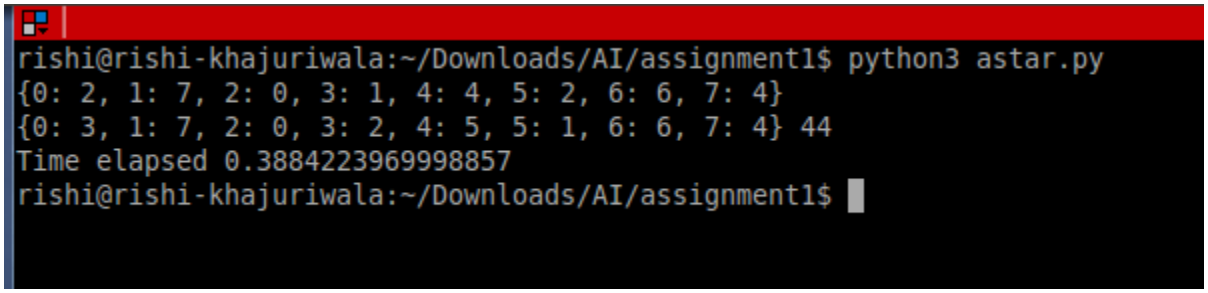
Greedy hill climbing for 31 Queens (time elapsed and the solved state)

A-star:

A-star solves a 8 queens problem very rarely (2 out of 20 times) within 10 seconds, for 7 queens problem it solves very rarely within 10 seconds as well (3 out of 10 times). It is better for a 6 Queens problem, solving it 6-7 times out of 10 times within 10 seconds. It works within one second for a 5 Queens problem. We would recommend trying out a 5 Queens problem just to ensure that you get a solution within the speculated time of 10 seconds. Sometimes even for a 7 Queens or an 8 Queens problem it runs for more than 10 minutes and we terminate the code.

```
rishi@rishi-khajuriwala:~/Downloads/AI/assignment1$ python3 astar.py
{0: 3, 1: 6, 2: 3, 3: 2, 4: 4, 5: 0, 6: 4}
{0: 3, 1: 6, 2: 4, 3: 1, 4: 5, 5: 0, 6: 2} 47
Time elapsed 0.4733073030001833
rishi@rishi-khajuriwala:~/Downloads/AI/assignment1$ python3 astar.py
{0: 3, 1: 0, 2: 2, 3: 0, 4: 3, 5: 6, 6: 3}
{0: 5, 1: 1, 2: 4, 3: 0, 4: 3, 5: 6, 6: 2} 50
Time elapsed 0.8690835119996336
rishi@rishi-khajuriwala:~/Downloads/AI/assignment1$ python3 astar.py
{0: 5, 1: 6, 2: 2, 3: 3, 4: 3, 5: 1, 6: 1}
{0: 4, 1: 6, 2: 1, 3: 3, 4: 5, 5: 0, 6: 2} 58
Time elapsed 4.924193187000128
rishi@rishi-khajuriwala:~/Downloads/AI/assignment1$
```

A star for 7 queens (solved state and Time elapsed)



A star for 8 queens (solved state and Time elapsed)

2) What is the effective branching factor of each approach? For this computation, perform 10 runs of a puzzle half the maximum size you calculated for step #1.

A-Star:

For finding the effective branching factor of A-star we are using a 5 Queens board. Over 10 runs we get the effective branching factor as 13.81. This is because after it goes up to a certain depth it then realizes that there is a state of the board with the better cost + heuristic, than the current state, at some level that it has already crossed. So, it backtracks and then expands that node. This keeps happening until the solution is popped out of the priority queue.

Hill Climbing:

The size of board used for this method is 15 Queens. Over 10 runs we get the effective branching factor as 1. This is because it does not keep track of all the states for which it calculates the heuristic unlike A-star. There is only one step that can reduce the heuristic of the board the most and it takes it. When there is no step possible to reduce the heuristic, it reaches a local minimum. Then we restart the board.

3) Which approach comes up with cheaper solution paths? Why?

A-star comes up with cheaper solution paths compared to Hill-Climbing. This is because A-star considers also the cost of the path taken and not just the heuristic, unlike Hill-Climbing that considers only the heuristic. This leads to the Hill-climbing approach taking costlier moves only because the heuristic is reducing. The Hill-Climbing might also get stuck at a local-minima and therefore is forced to restart.

4) Which approach typically takes less time?

The Hill-Climbing approach takes lesser time than the A-star approach. This is because A-star must check for the best cost solution. This results in the A-star exploring more nodes as we can see from the high branching factor we got in the second question. We can get A-star to reach a less optimal solution but quickly by increasing the weight of the heuristic.

Part 2. Urban planning

1) Explain how your genetic algorithm works. You must describe your selection, crossover, elitism, culling, and mutation approaches.

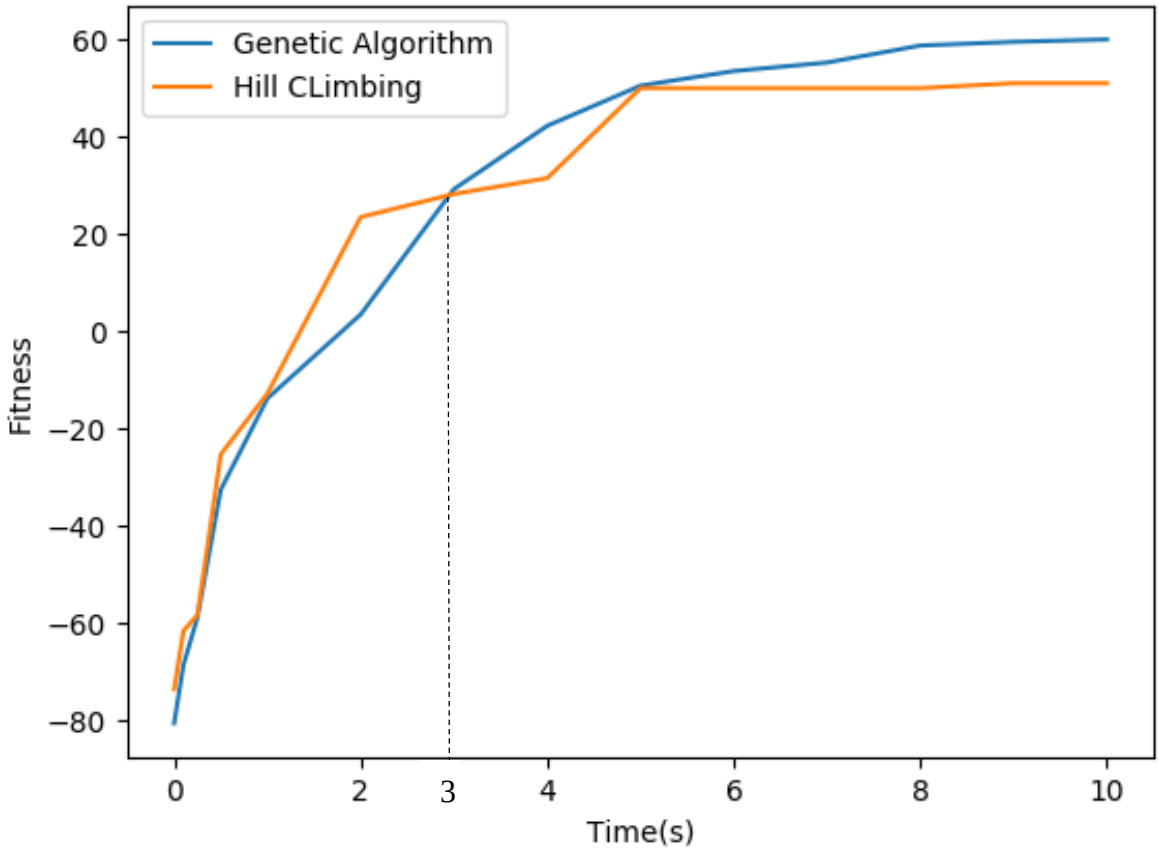
Our GA takes in the input map from the 'test' file and stores it in the variable named 'self.given_map' which is in the Map class. In our code, just for simplification of writing code, we have considered all the sites as an integer, i.e., "X"-Toxic Site as 10, "S"-Scenic Site as 11, "I"-Industrial as "12", "R"-Residential as "13", "C"-Commercial as 14. Then we have defined a method named "initialize_map" in which we have built the given number of industrial, residential and commercial sites in a duplicate map named "self.my_map". This duplicate map randomly has the positions of the I, R and C sites with multiple zeros. Then we made a list of 100 maps by permuting the duplicate map we had generated. This forms the population. We also make sure that I, R and C aren't built on the toxic sites. We then found the score for every member in the population. Now, for creating the next generation (offspring), we need to do cross-over of the parent members stored in the population array that is we need to select two parent members and mate them to create a new offspring which will have the qualities or the chromosomes (I, R, C) of the parent members. For the selection process of two parent members we used stochastic sampling method and mated them. After creating a population of offspring, we needed to carry out elitism to preserve the individuals having best score out of the population, so that it does not get lost while mating and we don't get a bad offspring. To remove the parent members with worst scores we need to do culling and to change the state of the population randomly we need to mutate population sometimes. In our code, we set aside the 10% of the maps (individuals-parents) with best scores from the population, append in an empty list (offspring) and keep them stored. For culling, we found the maps (parent members) with worst scores and removed them from the population array using the reverse priority queue method. The cross-over was done using the remaining un-culled 90% of the population.

In our crossover code, we have used random probabilities to determine whether to do the crossover of I, R or C of the parent maps to create a new offspring. So if the probability is below 0.35, the program will do the crossover of 'I' chromosome, if the probability is between 0.35 and 0.7 the program will do the crossover of 'R' chromosome and if the probability is above 0.7, the program will do the crossover of 'C' chromosome. We made sure the switch was a valid one by checking if there's no toxic or R or C if we are switching I's. Similarly for R and C. Mutation is also an important part in this because sometimes the algorithm for small maps gets stuck to a local maxima that is the maximum score in one generation so in order to diversify, we need to add some random maps or mutate maps in the generation. For mutation, we are taking 20% of the parent maps and randomly changing the positions of the I, R or C tile in the map, so as to diversify the score values. We again make sure if the switch does not lead to construction on a toxic site.

2) Create a graph of program performance vs. time. Run your program 10 times and plot how well hill climbing and genetic algorithms perform after 0.1, 0.25, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10 seconds. If you only had 0.25 seconds to make a decision, which technique would you use? Does your answer change if you have 10 seconds?

The graphs shown below show the performance of the genetic algorithm and hill climbing on a map of size 10x10. For smaller maps like sample 1, Hill climbing is incredibly fast to reach a high score. The graph was basically a straight line parallel to X-axis at the max score. The starting score for both the methods are slightly different because they were run separately. This is not a problem as we want to see the average performance of both the algorithms. The graph shows the best fitness of the population

achieved at that instant. A few places it keeps increasing as the map is still undergoing changes. The end portion of the map from 6 to 10 seconds is almost a straight line parallel to x-axis because the Hill Climbing is not able to find a better fitness solution than the current best one even with restarts. We can see that Genetic Algorithm does not allow the score to decrease (monotonical increase) due to elitism. For small maps we feel that Hill climbing is better than Genetic Algorithm and for large maps Genetic Algorithm is better. If you want a solution in a very small amount of time we suggest using Hill Climbing over Genetic Algorithm for the same starting state, as the solution given by Hill Climbing would have a better fitness than that of Genetic Algorithm (Clearly visible in the graph). In Genetic Algorithm the fitness achieved after a long time is much better than the best solution achieved by Hill Climbing for that same amount of time. We can see that in the graphs as well.

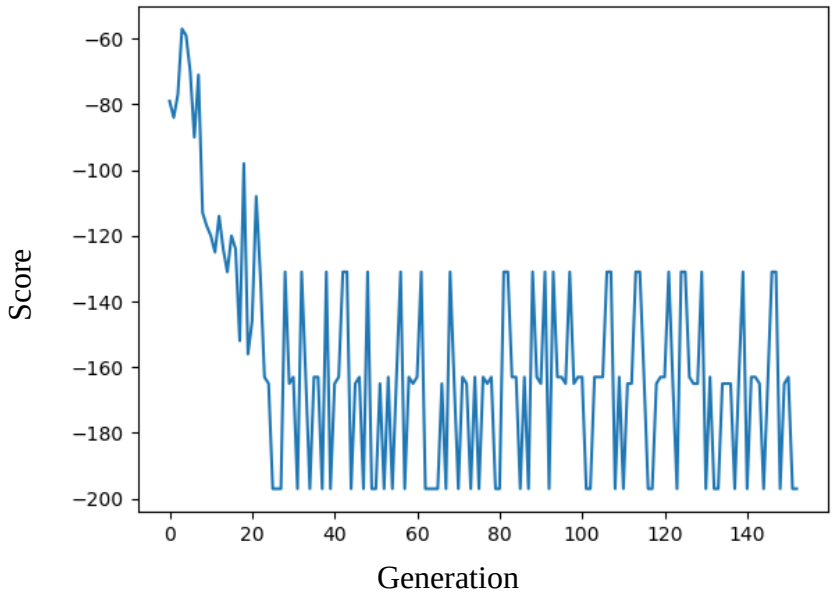


Genetic Algorithm v/s Hill Climbing

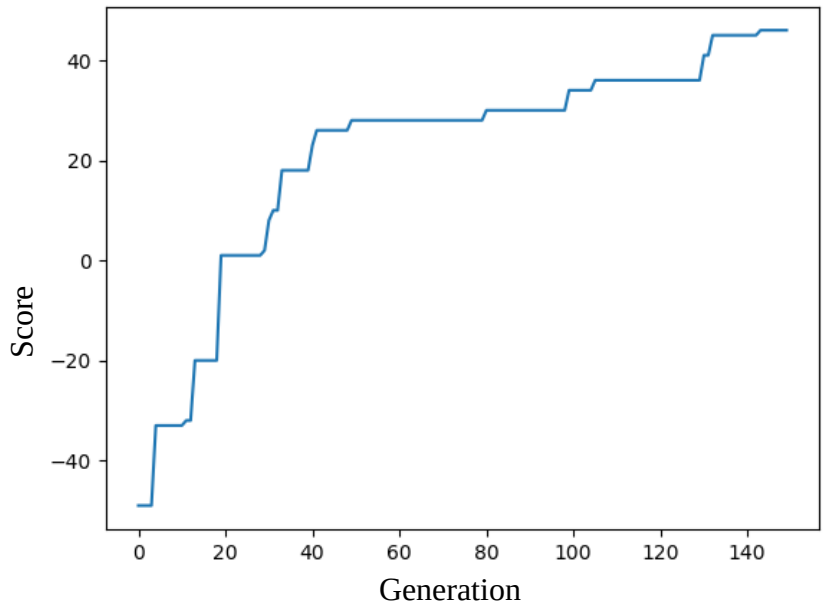
We can see that after 3 seconds the Genetic Algorithm starts performing better than the Hill Climbing. And also that the final score achieved by Genetic Algorithm is better than that achieved by Hill Climbing. Genetic Algorithm is slow to start but as time passes gives better solutions than Hill Climbing with restarts.

3) How do elitism and culling affect performance?

Without elitism there is a chance for the best individuals in a population being lost in the next generation. This results in the maximum score of the population to vary and not monotonically increase. Culling ensures we don't select the percentage of population that are not suitable for producing offspring by removing them from the population. Elitism creates a copy of the best parent individuals in the offspring population. We can see the performance in the below graphs.



Without Culling and Elitism



With Culling and Elitism (Monotonic Performance)

4. How did you perform selection and crossover for your population? Find some known method for doing selection, other than the one described in class. Explain what you did.

We used a stochastic sampling method for the selection of individuals to mate, i.e., to crossover. One other method for selection we found was choosing one parent from the top 50% of the population and another parent from the entire population. This is also proven to give rise to good offspring. The method discussed in class was to give weightage to selection based on the fitness.
For crossover, we have used random probabilities to determine whether to do the crossover of I, R or C of the parent maps to create a new offspring. So, if the probability is below 0.35, the program will do the crossover of 'I' chromosome, if the probability is between 0.35 and 0.7 the program will do the crossover of 'R' chromosome and if the probability is above 0.7, the program will do the crossover of 'C' chromosome. We made sure the switch was a valid one by checking if there's no toxic or R or C if we are switching I's. Similarly, for R and C.