T.C

MANİSA CELÂL BAYAR UNIVERSITY

ENGINERRING FACULTY

COMPUTER ENGINEERING DEPARTMENT

ARTIFICIAL INTELLIGENCE

AI – PA4 Report

Prepared by

Berkay Şahin 170316012
Ayşenur Büyükbal 170316007
Mustafa Deniz Demir 180316043

Instructor

Dr. Zeynep ÇİPİLOĞLU YILDIZ

FALL 2021-2022

TABLE OF CONTENT

Development Environment

Implementation machine's operating system is **Windows** 10 Home Pro. We used **Microsoft Visual Studio Code** environment to implement **Simple AI Library**'s abstract searching problem class to our N-Queens Problem and **Python version 3.8.12** is used.

Problem Formulation

This problem is to find an arrangement of N-Queens on a chess board, such that no queen can attack any other queens on the board. The chess queens can attack in any direction as horizontal, vertical, horizontal, and diagonal way. But since we assume that each column has only one queen, so we do not need to check our vertical violation cases.

We keep our current arrangement state in a string which has length N and consists only integers between 1,…, N.

In this assignment, we are expected to formulate the N-Queens problem as a search problem and solve it using following local search algorithms:

- Hill-Climbing Search
- Random Restarts Hill-Climbing
- Genetic Search

Results

We have imported new search functions to our code from another file which is **`local.py`**.

To implement these functions, we must describe some other functions in models.py file. They were:

`value(self, state)` Fitness value, $C(n, 2) - attacking\ pairs$

`crossover(self, state1, state2)` Crossover method, choose middle point of state and use it crossover point.

`mutate(self, state)` Mutation method, generate random position and random value, and change the random position located queen's value with random value.

`generate_random_state(self)` Generating random state with N value.

Here are the different outputs for different parameters.

Hill-Climbing Search:

Iteration limit* = 0

**If iterations limit is specified, the algorithm will end after that number of iterations. Else, it will continue until it can't find a better node than the current one.*

```
Solution Method: Hill Climbing
Iteration Limit = 0 (default)

Solution time:
0.2750 seconds
Resulting state:
31425678
Viewer stats:
{'max_fringe_size': 1, 'visited_nodes': 0, 'iterations': 7}
```

```
[36]: instance = NQueens(8)

      Press '1' to set and '2' to generate randomly.
      1.Set state manually
      2.Set state randomly 1
      Enter your state 31425678
      Valid State

[37]: instance.count_attacking_pairs()

[37]: 6
```

Iteration limit = 5

```
Solution Method: Hill Climbing
Iteration Limit = 5

Solution time:
0.1570 seconds
Resulting state:
36425678
Viewer stats:
{'max_fringe_size': 1, 'visited_nodes': 0, 'iterations': 5}
```

```
[46]: instance = NQueens(8)

      Press '1' to set and '2' to generate randomly.
      1.Set state manually
      2.Set state randomly 1
      Enter your state 36425678
      Valid State

[47]: instance.count_attacking_pairs()

[47]: 7
```

Iteration limit = 350

```
Solution Method: Hill Climbing
Iteration Limit = 350

Solution time:
0.3309 seconds
Resulting state:
31425678
Viewer stats:
{'max_fringe_size': 1, 'visited_nodes': 0, 'iterations': 7}
```

Same output with Iteration limit = 0

```
[36]: instance = NQueens(8)

      Press '1' to set and '2' to generate randomly.
      1.Set state manually
      2.Set state randomly 1
      Enter your state 31425678
      Valid State

[37]: instance.count_attacking_pairs()

[37]: 6
```

Random Restart Hill-Climbing Search:

Restarts Limit* = 5

* *Restarts Limit specifies the number of times Hill Climbing will be run.*

```
Solution Method: Hill Climbing - Random Restarts
Restarts Limit = 5

Solution time:
2.1553 seconds
Resulting state:
63741125
Viewer stats:
{'max_fringe_size': 1, 'visited_nodes': 0, 'iterations': 16}
```

```
[24]: instance = NQueens(8)

      Press '1' to set and '2' to generate randomly.
      1.Set state manually
      2.Set state randomly 1
      Enter your state 63741125
      Valid State

[25]: instance.count_attacking_pairs()

[25]: 3
```

```
Solution Method: Hill Climbing - Random Restarts
Restarts Limit = 30

Solution time:
5.3082 seconds
Resulting state:
46827175
Viewer stats:
{'max_fringe_size': 1, 'visited_nodes': 0, 'iterations': 94}
```

```
[30]: instance = NQueens(8)

      Press '1' to set and '2' to generate randomly.
      1.Set state manually
      2.Set state randomly 1
      Enter your state 46827175
      Valid State

[31]: instance.count_attacking_pairs()

[31]: 1
```

```
Solution Method: Hill Climbing - Random Restarts
Restarts Limit = 75

Solution time:
9.7145 seconds
Resulting state:
26831471
Viewer stats:
{'max_fringe_size': 1, 'visited_nodes': 0, 'iterations': 256}
```

```
[26]: instance = NQueens(8)

      Press '1' to set and '2' to generate randomly.
      1.Set state manually
      2.Set state randomly 1
      Enter your state 26831471
      Valid State

[27]: instance.count_attacking_pairs()

[27]: 1
```

```
Solution Method: Hill Climbing - Random Restarts
Restarts Limit = 125

Solution time:
16.4646 seconds
Resulting state:
66314752
Viewer stats:
{'max_fringe_size': 1, 'visited_nodes': 0, 'iterations': 406}
```

```
[28]: instance = NQueens(8)

      Press '1' to set and '2' to generate randomly.
      1.Set state manually
      2.Set state randomly 1
      Enter your state 66314752
      Valid State

[29]: instance.count_attacking_pairs()

[29]: 1
```

```
Solution Method: Hill Climbing - Random Restarts
Restarts Limit = 200

Solution time:
25.9020 seconds
Resulting state:
84136275
Viewer stats:
{'max_fringe_size': 1, 'visited_nodes': 0, 'iterations': 678}
```

```
[32]: instance = NQueens(8)

      Press '1' to set and '2' to generate randomly.
      1.Set state manually
      2.Set state randomly 1
      Enter your state 84136275
      Valid State

[33]: instance.count_attacking_pairs()

[33]: 0
```

```
Solution Method: Hill Climbing - Random Restarts
Restarts Limit =200

Solution time:
26.4240 seconds
Resulting state:
81473635
Viewer stats:
{'max_fringe_size': 1, 'visited_nodes': 0, 'iterations': 672}
```

```
[34]: instance = NQueens(8)

      Press '1' to set and '2' to generate randomly.
      1.Set state manually
      2.Set state randomly 1
      Enter your state 81473635
      Valid State

[35]: instance.count_attacking_pairs()

[35]: 1
```

Genetic Search:

```
Solution Method: Genetic
Population Size: 5
Mutation Chance: 0.1

Solution time:
0.0010 seconds
Resulting state:
86663153
Viewer stats:
{'max_fringe_size': 5, 'visited_nodes': 0, 'iterations': 1}
```

```
[44]: instance = NQueens(8)

      Press '1' to set and '2' to generate randomly.
      1.Set state manually
      2.Set state randomly 1
      Enter your state 86663153
      Valid State

[45]: instance.count_attacking_pairs()

[45]: 8
```

```
Solution Method: Genetic
Population Size: 10
Mutation Chance: 0.1

Solution time:
0.0036 seconds
Resulting state:
82642537
Viewer stats:
{'max_fringe_size': 10, 'visited_nodes': 0, 'iterations': 1}
```

```
[50]: instance = NQueens(8)

      Press '1' to set and '2' to generate randomly.
      1.Set state manually
      2.Set state randomly 1
      Enter your state 82642537
      Valid State

[51]: instance.count_attacking_pairs()

[51]: 4
```

```
Solution Method: Genetic
Population Size: 20
Mutation Chance: 0.1

Solution time:
0.0076 seconds
Resulting state:
83722514
Viewer stats:
{'max_fringe_size': 20, 'visited_nodes': 0, 'iterations': 2}
```

```
[42]: instance = NQueens(8)

      Press '1' to set and '2' to generate randomly.
      1.Set state manually
      2.Set state randomly 1
      Enter your state 83722514
      Valid State

[43]: instance.count_attacking_pairs()

[43]: 1
```

```
Solution Method: Genetic
Population Size: 100
Mutation Chance: 0.1

Solution time:
0.0250 seconds
Resulting state:
82747165
Viewer stats:
{'max_fringe_size': 100, 'visited_nodes': 0, 'iterations': 1}
```

```
[40]: instance = NQueens(8)

      Press '1' to set and '2' to generate randomly.
      1.Set state manually
      2.Set state randomly 1
      Enter your state 82747165
      Valid State

[41]: instance.count_attacking_pairs()

[41]: 3
```

```
Solution Method: Genetic
Population Size: 200
Mutation Chance: 0.1

Solution time:
0.0412 seconds
Resulting state:
81372734
Viewer stats:
{'max_fringe_size': 200, 'visited_nodes': 0, 'iterations': 1}
```

```
[38]: instance = NQueens(8)

      Press '1' to set and '2' to generate randomly.
      1.Set state manually
      2.Set state randomly 1
      Enter your state 81372734
      Valid State

[39]: instance.count_attacking_pairs()

[39]: 3
```

Discussion

Hill Climbing Search

	Iterations Limit	Attacking pairs	Time(s)
Hill Climbing Search	0	6	0.2750
	5	7	0.1570
	350	6	0.3309

This set of algorithms (local searches) works very fast. But unfortunately, it does not guarantee a solution. We can say that the Hill Climbing algorithm gives results far from the solution. The Hill Climbing algorithm, by its nature, gives us the best result it can find with optimizations over only one state. This "best" result will vary depending on where the algorithm starts its search. Therefore, we can say that we have a limited scanning area in this algorithm. In addition, we have the possibility of getting stuck at some special points (local maximum, local minimum) in the state space of our algorithm. Just looking at the table, we can say that the iterations limit has almost no effect on the result, it only prolongs the time. Of course, all these analyzes are valid only for this problem definition and the parameters we give. When the algorithm runs unlimited (limit = 0), it does 7 iterations, and returns a state with 6 attacking pairs. When we set the iterations limit to 5, the algorithm moves a little further from the solution and returns a state with 7 attacking pairs. Because algorithm couldn’t reach remaining 2 iterations since limit is 5. When the limit is set to 350 again, the algorithm behaves as if there is **no limit** and returns with a state with 6 attacking pairs same as the first case.

Random Restarts Hill-Climbing Search

	Restarts Limit	Attacking pairs	Time(s)
Random Restarts Hill-Climbing Search	5	3	2.1553
	30	1	5.3082
	75	1	9.7145
	125	1	16.4646
	200	0	25.9020
	200	1	26.4240

When we look at the table, we can see that we achieved much better results than the previous algorithm even with the lowest restarts limit value. However, it should be noted that this time we are not as fast as the previous algorithm in terms of time.

As we increase our limit, we see that we get results very close to the solution, but we cannot obtain the solution even though the increase continues. In all cases up to limit = 200 the solution is not exact but close.

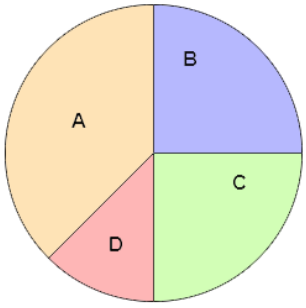
When we set our limit value to 200, we get a different result. This time, there are two different situations for the value of 200: in the first, we get a complete solution, while in the other, we get a result that is very close to the solution, like the others. So, this means that as our limit value increases, we get closer to the result, and we can even get an exact solution, but this is not certain.

Genetic Search

	Population Size	Attacking pairs	Time(s)
Genetic Search	5	8	0.0010
	10	4	0.0036
	20	1	0.0076
	100	3	0.0250
	200	3	0.0412

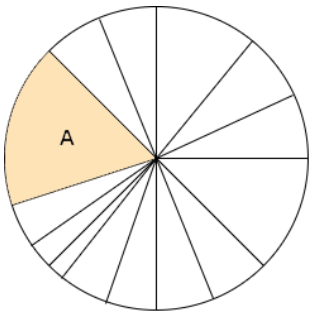
Mutation chance = 0.1

We can say that genetic search algorithms are the fastest algorithms ever. However, the small size of the population negatively affects the quality of the result. When we give the population size 5, a state containing 8 attacking pairs is returned as the closest solution. This result is, of course, quite far from the real solution. We got better results when we increase the population size by one more level, but it's still not enough. On the one hand, our time value increases in direct proportion to the population size. When the population was 20 to you, we got the closest result to the solution. Despite running it over and over, we've never gotten a solution closer to this size than this. Interestingly, we moved away from the solution when we increased the population size to see how the outcome would change. We **think** that the larger the population size, the less likely it is that individuals with a high fitting value will be selected.



Population size : 5

Individual A has the biggest chance to be selected.



Population size : 15

Still biggest chance but too many other possibilities