

CS420

Project 2

N Queens

Junda Lou

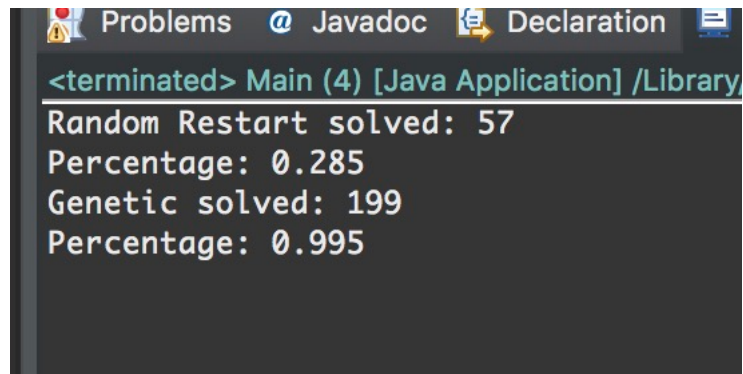
Local Search turns out to have good effects on the N-Queens problem. When I first implemented the algorithms. I did not think too much, and just went for a 2-D array for the board, which is 21×21 in this case. Then I found it was really time and space consuming, and it did a lot of redundant steps to check number of attacking pairs. In fact, no 2 queens will be placed on the same row or column due to the rules, so an 1-D array with n cells is enough. It guarantees that no 2 queens will be in the same column by default.

Implementing Random Restart Hill Climbing, I use a priority queue for storing all the neighbors of the current states, which the initial state is randomly generated with a maximum of 25 retries. The comparator for the queue is the number of attacking pairs, the less, the better. If no neighbor is better than the current state, then it is done, otherwise pop the head of the queue, which is the available best results, and repeat the process.

Implementing Genetic Algorithm, I also use a priority queue with the same comparison function. Parents are selected from the population based on their "fitness". Although it is randomly chosen, better fitness will result in better chance to be chosen. A child is totally randomly produced using 2 parents selected, and it has a 10% chance to mutate, meaning to modify a certain queen to another position. Repeating the process, until the number of attacking pairs is 0.

Genetic Algorithm is more efficient than Random Restart Hill Climbing based on my results. Random Restart Hill Climbing returns a local minima every time. Assume there are n minima for the problem, n restarts should have a global minima as a result. In this case, we have 25 restarts, and my results show an average percentage of 35% is solved, so there is approximately $25/0.35 = 71$ minima in the 21-Queens problem. For the Genetic Algorithm, it is more random on modifying the board, that is, random choosing 2 parents, random producing a child, and random mutate a child. The size of the population plays a big role here. Too large of the size will significantly slow the program down, while too small of the size will make the algorithm slow find the answer. 90 is the best size based on my tests. Mutation probability is another important fact to consider here. Based on my tests, 10% performs the best. I do not

know if there is a scientific way to find the best number for these parameters, but have a great amount of tests is a good way to practice.



```
<terminated> Main (4) [Java Application] /Library
Random Restart solved: 57
Percentage: 0.285
Genetic solved: 199
Percentage: 0.995
```

Genetic Algorithm Time Limit = 1 sec per run here. Different limit gives different results.

In general, I find Genetic Algorithm is much more efficient than Random Restart Hill Climbing.