California Polytechnic State University, Pomona
CS420 Section 1
Artificial Intelligence

Jacob Lepere
ID: 012839151

Project 2: 21-Queens
02.13.2018

Introduction

The problem addressed is that of placing 21 queens on a 21x21 chess board such that no queens are attacking each other. Per standard chess rules, a queen can move vertical, horizontal, or diagonal along the board. Due to the size and complexity of the problem, artificial intelligence search techniques were implemented to find a solution.

Local search techniques were chosen to solve the problem to mitigate the space complexity overhead. Since we keep one or a fixed number of states in memory, we greatly reduce the search space for the problem. The two algorithms implemented were Random Restart Hill Climb (RRHC) and the Genetic Algorithm (GEN). Although both algorithms fall under the local search category, they are extremely different in their implementation.

RRHC is a variation of the standard local search algorithm, Hill Climbing (HC). HC has one state in memory and modifies this state by swapping it with the best (highest objective function value) successor of that state. The algorithm terminates when no successors of the current state are better than the current state. RRHC uses HC continuously, terminating when a solution is returned and restarting from a random initial state otherwise, hence the name Random Restart Hill Climb.

Rather than one current state, GEN has n states in memory, also known as the population. To generate the next population, parents from the current population are selected for breeding. The resulting child state is then mutated and added to the new population. After replacing the current population with the new population, the algorithm tests if an individual in the population is fit enough. If this is the case, the algorithm terminates. Otherwise it continues with the next generation. The correlation between GEN and Darwinian theory is easily recognizable.

Approach

The state space was drastically reduced by inferring that a solution must have one and only one queen in each row. Therefore, a state was chosen to be a 21-size array of integers where index i of the array represented the column the queen in row i was in. For example, [0, 1, 2, …, 20] represented the queens on the diagonal, from top left to bottom right. The objective function or fitness score for RRHC and GEN, respectively, was the max number of conflicts (a queen in the path of another queen) minus the number of conflicts of the state. The max number of conflicts is $\binom{21}{2} = 210$.

RRHC was implemented using at most 25 iterations of HC. GEN is not so straight forward. First, the population size was chosen to be 100, the mutate probability 1/21 = 0.048, and the maximum number of generations 5000. These were chosen by personal feel for generating the highest percentage of solutions in a practical amount of time. Now, the complex components for GEN are the selection phase, the breeding phase and the mutations phase. An in-depth reasoning for why the following methods were chosen is included in the conclusion of this report. The individual with the highest fitness score was given a 50% chance to be selected as a parent, the second highest a 25% chance, the third a 12.5% chance, and so forth. Breeding was done by selecting a random index between (21*(1/4), 21*(3/4)) = (5, 15) inclusive for the lower bound and exclusive the upper bound. Then, the child would be given (0, i-1) queen positions from the first parents and (i, 20) queen positions from the second parents, from the selected index i. Mutation was done by iterating each queen position and moving that queen position to a random location anywhere on that row with probability 0.048.

Findings

Row Labels	Executions	Average(SEARCH_COST)	Average(TIME [ms])
GEN	1000	238146	356.62
FALSE	664	256241	383.61
TRUE	336	202386	303.28
RRHC	1000	21981	140.28
FALSE	597	4095	175.44
TRUE	403	48476	88.20
Grand Total	**2000**	**130063**	**248.45**

The chart above is a consolidation of the results of executing RRHC and GEN 1000 times, each. Whether the result was a solution (True for yes and False for no), the search cost (number of nodes generated at any point in the algorithm), and the time to run was computed for each execution. Although it is hard to compare the two algorithms because there is an infinite number of configurations of GEN and the correlation between max iterations for RRHC and max generations for GEN is low, there are still inferences we can make from the results.

First, it is easy to see that RRHC executed more than twice as fast as GEN, on average. Also, RRHC found solutions roughly 40% of the time compared to 34% for GEN. As expected, the average run time for executions that did not generate a solution were longer than executions were a solution was found. What is interesting, however, is that the search cost for RRHC solutions found executions is an order of magnitude higher than those executions when a solution is not found. An explanation for this phenomenon could be that local maxima are found very quickly, while solution paths are quite long. Although this justifiable, it may not be the entire story because plateau or shoulder walks were not allowed. Also, non-solution runs execute HC 25 times while solution runs execution HC between 1 and 25 times. More analysis could be done in this regard.

Conclusion

This project enforced understanding of two local search algorithms, Random Restart Hill Climb and the Genetic Algorithm. A common queen positioning problem was assigned to test these algorithms. Results were found that aligned with initial thoughts, while others were less intuitive. Now, it is important to reflect on some of the problems faced during implementation. Following is a list of the most prevalent and frustrating issues.

- Conventional Genetics Algorithms use a bit representation for the states (1001010…). This made it complicated to breed and mutate on a bit level because it could lead to queen position representations outside the board since 21 numbers take 5 bits to represent. If a queen position is 00111 (7) and the most significant bit is mutated, the result will be 10111 (23) which is out of the bounds of the board.
- String processing was slow, another reason why integer arrays were used.
- Remembering fitness score of an individual and greatest fitness score of the population was important for efficiency.

- After a few iterations of GEN, the population was very similar in fitness scores. Therefore, giving the traditional priority of (fitness score / total fitness score) to everyone for parent selection did not give a great enough priority to higher fitness scored individual.
- Selecting optimal population size and mutate probability.

An optimistic programmer would believe that the more bugs you have the more opportunities you have to learn.

In the future, different configurations of GEN could be analyzed to find the one that outputs the highest percentage of solutions. Maybe GEN can be used to find the most optimal GEN configuration.