Michael Trinh

CS 4200

Professor Atanasio

Project 2: N-Queen

Introduction

In project 2, I implemented two algorithms that can solve the N-Queen problem. One being simulated annealing and the other being genetic algorithm.

For simulated annealing, I generated a random board of n queens and slowly solved the board by randomly rearranging a queen's position to another row of that same column, and having a decaying acceptance rate to choose whether or not to accept a downhill move. There is a limit 4000*nqueens iterations (100000 for 25 queens) the algorithm will perform, if unsolved, before it stops.

For genetic algorithm, I generate a population of randomly generated n queen boards to be used for a single problem. Each board get its fitness evaluated by the number of non-attacking queen pairs, and has the parent successors chosen probabilistically by the fitness of the boards for that population. Once successors are chosen, a child board is created by crossing over the two successors using a random crossover point. The child can probabilistically mutate one of its queen rows to a random one. This is done until the amount of child boards is the same as population. The child boards are now the new parents; we check if any of them are solutions; then repeat starting with getting fitness. The limit I used here was 100000; the population is 10*nqueens with a 5% mutation rate.

Analysis

I tested the Simulated Annealing algorithm each with 1000 iterations using 25 nqueens.

For Genetic Algorithm, I only tested 10 iterations due to its slow computation.

Data:

Algorithm	% of solved problems	Average Search Cost	Average Runtime
Simulated Annealing	98.2%	29437 iterations	0.079128 seconds
Genetic Algorithm	100.0%	27163 generations	637.92532 seconds

Simulated Annealing runtime is much faster than genetic algorithm because GA has more overhead than SA due to having to created n population children and checking if those children are a solution as well as evaluating their fitness. SA isn't guaranteed to solve the problem but GA is because I tested GA only with 10 iterations and that SA has a limit of 100000 iterations for 25 queens, meaning that some problems that aren't solved within 100000 iterations are unsolved. If n iterations were infinite, both algorithms are guaranteed to be solved. If GA was tested with 1000 iterations, % of solved problems will most likely decrease. Average search costs for GA is lower than SA because GA uses n population states to try to solve, while SA uses a population of only 1 state, which means GA will, on average, have a lower search cost than SA.

Findings 2613

Overall, SA is faster than GA in my version of the implementation. Even though GA has a lower search cost, which is number of generations, compared to SA's search cost, which is number of iterations done on the problem. Implementation of SA wasn't too difficult, but implementing and optimizing GA was very difficult due to having to test multiple mutation rates, population sizes, and different ways to lower overhead. My implementation of GA is very slow, but gives a solution. A way to improve my GA implementation could be to use a 1 dimensional string, in which each character represent a queen's x-position and index to represent y-position to represent a problem in my population.