Solving the N-Queens Problem Using a Genetic Algorithm

Dugan Dobbs[1] and Jon Shaak[1]

[1]Department of Computer Science, Texas A&M University, Corpus Christi, TX 78412, USA

ABSTRACT

In this paper, a genetic algorithm is applied to the N-Queens problem to find all possible solutions for a particular N. A solution for the N-Queens problem is defined as a board state in which no queens threaten any other. This means they cannot be in the same diagonal, row, or column as any other queen. The genetic algorithm will use the principles of population, selection, crossover, and mutation which will be covered in more detail in this paper.

1 INTRODUCTION

The N-Queens problem began with the 8-Queens puzzle first introduced by chess composer Max Bezzel in 1848. In 1850, Franz Nauck proposed the problem in an issue of Illustrirte Zeitung, a famous German newspaper, and originally asserted that there were 60 solutions to the 8-queens problem (cite here 1).The famous mathematician Carl Friedrich Gauss proposed 76 solutions right before Nauck corrected himself and said there were 92 solutions (cite here 1). The important thing to note here is that it took a couple of years to figure out the total number of solutions. To think of how difficult it was to brute force this problem, there are a total of 4.4 billion possible arrangements of 8 queens on an 8 x 8 chessboard. Now, it is proven that if one can find the fundamental solutions of the problem, then the total number of distinct solutions can be found. For example, the 8 queens problem has 12 fundamental solutions and 80 secondary solutions. It is important to recognize that fundamental solutions are distinct solutions that are not rotations or reflections of each other. Based on this fact, we split these 12 fundamental solutions into rotations and reflections. The first solution can be split into four distinct solutions including itself and its rotations. With the 11 remaining states, it is taken into account their four rotations and reflections. Therefore the solution for the number of states yields 11*8 + 1*4 = 92 total distinct states.

Now, over a century later, the N-queens problem has become a common puzzle in the field of artificial intelligence, with the first algorithm for solving this problem coming from Edsger Dijkstra in 1972. He demonstrated the power of structured programming, demonstrating his depth-first backtracking algorithm to solve the N-queens problem. While this algorithm is not the most efficient way to solve this problem, it introduced the N-queens problem into the field of artificial intelligence.

2 PROBLEM DEFINITION

The problem states to find the number of solutions, the number of iterations, and the time elapsed to find all solutions of the N-Queens problem using a genetic algorithm. The goal of this study is to prove the efficacy of using a genetic algorithm over other existing algorithms that are used to solve the N-queens problem such as the backtracking algorithm published by Dijkstra. The algorithm used in this study will use a constant population of board states that are randomly generated, a fitness function utilizing the maximum number of attacks a state can achieve as well as the current number of attacks, a tournament selection algorithm paired with a selection probability, a three-way tournament crossover algorithm, and single mutation. Exactly how they are implemented will be further elaborated in the equations and implementations section. The experiment will attempt to limit the iterations and time it takes to find all solutions for any given N.

3 APPROACH

3.1 Genetic Algorithms

A genetic algorithm is a search heuristic inspired by the natural selection theorem of Charles Darwin.It utilizes the main principles of natural selection; those principles being population, selection by fitness, genetic crossover, and genetic mutation. The principles suggest that fit offspring be produced to survive in the environment they are placed in. In the case of a genetic algorithm, fit offspring are created in each generation until the most fit individual in a population is found, a goal state. In the 1960s, John Holland and his colleagues invented genetic algorithms to attempt to formally study the phenomenon of adaptation as it occurs in nature (cite here 2). The other goal of creating a genetic algorithm was to study the mechanisms of natural adaptation and how it could be imported into computer systems (cite here 2). This was a major innovation as many problems in computer programming require programs to be adaptive to changing conditions.

With problem solving with computers becoming more sophisticated, computer scientists are starting to have a hard time programming problems algorithmically, meaning there are heuristics that are too difficult to compensate for. However, with the innovation of genetic algorithms, the algorithm can adapt to constantly changing environments by creating new ”child” states that have acquired better genes than its parents to survive and reproduce until the goal is found.

There are five phases of a genetic algorithm, the population, fitness function, selection, crossover, and mutation.

3.1.1 Population

The population is defined by the program designer and should be an optimum amount. If the population is too small, the algorithm will take too much time finding solutions. If the population is too large, the algorithm will also take too long to determine which states are fit to reproduce.

3.1.2 Fitness

The fitness function is determined by the programmer as well depending on the problem they are trying to solve. The function should use a value that represents the most unfit individual state possible and compare the current state’s fitness value with the most unfit. The greater the difference between the current state’s fitness and the most unfit state’s fitness, the more fit the current state actually is.

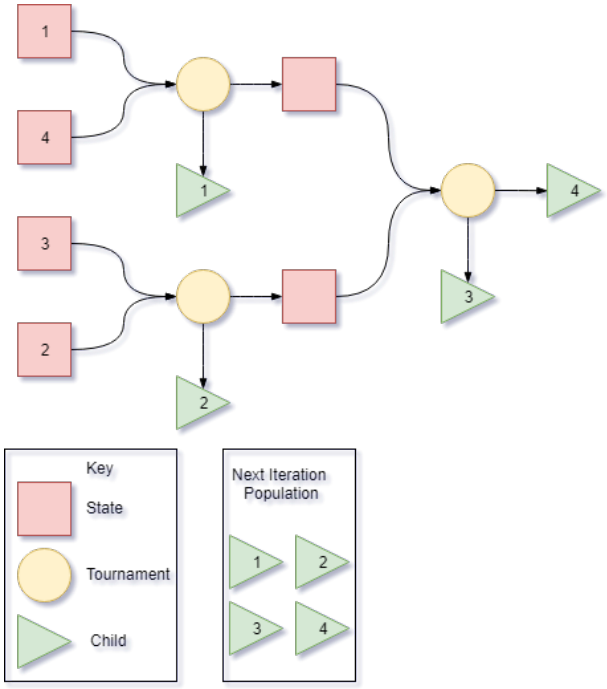

Figure 1: Sample Tournament With Population of 4

Figure 2: There is supposed to be a board here...

3.1.3 Selection

Selection determines a probability threshold that determines a state to be selected or not. If the state's fitness does not exceed the probability threshold, it will not be selected as a parent to produce offspring for the next generation.

3.1.4 Crossover

Crossover determines what alleles will be passed on to the child. The child will inherit its alleles from the parent, creating a child with a combination of the two parents, but not being the same as either parent.

3.1.5 Mutation

Mutation occurs when a random probability the child will mutate matches or exceeds the probability that it will mutate. When this probability is met, the child will have some of its alleles mutate at random that could potentially make it a stronger or weaker state.

3.2 March Madness Selection-Tournament

The proposed March Madness selection tournament is inspired by the March Madness basketball tournament structure. This is a single elimination bracket that start with a power of two number of teams, and ends with one winner. A simple four team bracket example is included in figure 3.2. To begin with, the teams are sorted into best-worst pairings. This means that the teams are sorted by fitness, then they are paired with their compliments at the end. In an array structure, the first tier of tournaments are run with teams [1] and [4], then teams [2] and [3]. In terms of n population size, this could be viewed as the psudocode given in algorithms 1-3

Algorithm 1 Tournament Setup

```
procedure MARCHMADNESSRUNNER(n, bSize)
    populationSize ← 2^x
    solutions ← []
    population ← generatePops(x)
    while size(solutions) < totalSolutions do
        fitness ← GenerateFitness(n, bSize, population)
        sortedPairs ← PairAndSort(n, population, fitness)
        for x ← 0; x < populationSize; x++ do
            if sortedPairs[x].fitness == C(bSize, 2) then
                append(solutions, sortedPairs[x].pop)
            else
                break
        population ← MMTournament(sortedPairs)
```

Algorithm 2 Fitness Generation

```
procedure FITNESSGENERATION(n, bSize, population)
    fitness ← []
    for x ← 0; x < 2^n; x++ do
        popFit ← C(bSize, 2) − generateClash(population[x])
        append(fitness, popFit)
    return fitness
```

Algorithm 3 March Madness Tournament

```
procedure MMTOURNAMENT(n, bSize, population)
    while size(pop) > 1 do
        n ← size(pop)
        i ← n / 2
        for x in range(i) do
            t1 ← pop[x].
            t2 ← pop[n − x].
            child, winner_x ← Tournament(t1, t2)
            children_x ← SingleMutation(child)
        pop ← winner
    children_−1 ← pop
    return children
```

4 Related Works

A study performed by Uddalok Sarkar and Sayan Nag uses an adaptive genetic algorithm to solve the n-queens problem. They utilize the same principles as stated in the introduction in their genetic algorithm. They use the same fitness function and similar selection of population than the algorithm that will be discussed in this paper, but the elegant step they take is comparing the offspring to the current population and determining if the offspring is fit enough to survive the next generation. If it is not, the offspring is discarded so that its genetics are not added to the gene pool. The results of their algorithm showed that the algorithm only need to perform 1431 iterations to find all solutions for an N of 25.

[1] This is a garbage citation, it prevents LaTeX from exploding.

5 Results

Starting to copy paste results in, formatting comes later.

6 Discussion

A conclusion section is not required. Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions.

APPENDIX

Appendixes should appear before the acknowledgment.

ACKNOWLEDGMENT

References are important to the reader; therefore, each citation must be complete and correct. If at all possible, references should be commonly available publications.

References

[1] U. Sarkar and S. Nag. An Adaptive Genetic Algorithm for Solving N-Queens Problem. 2017.