Heuristic Search Practices. Course 2020-2021. N-queens with A* and Genetic Algorithms.

Hugo Fonseca Díaz
uo258318@uniovi.es

School of Computing Engineering. University of Oviedo.

Abstract. This paper conducts a theoretical and experimental research in order to solve the N-queens problem with two algorithms, the A* algorithm and a genetic algorithm. The first part of the document consists in a formal definition of the problem and of both algorithms, as well as how they are adapted in order to solve this particular N-queens problem. Data is then extracted from several executions of both algorithms and an analysis of the results is carried out. The paper ends with a comparison of both approaches and a small conclusion summing up everything that it covered.

Keywords: A* · Genetic algorithms · N-queens problem · Heuristic Search · Metaheuristics.

1 Introduction

This paper solves the popular problem known as *N-queens* by means of two different algorithms. These algorithms are the A* algorithm and a genetic algorithm.

The motivation for solving the N-Queens is to see how these algorithms behave against a problem with so many combinations that it requires a big computational calculation. The data obtained in this process is then used as basis for comparing both algorithms.

In the first part of the document a theoretical definition of the problem and of the algorithms will be carried out. After that, a description of the steps taken into adapting these algorithms for being able to solve the problem is given. Then, an experimental study will take place in order to measure the overall performance and effectiveness of the two approaches as well as a comparison between the results. Finally, the conclusions of the paper are presented.

2 N-queens Problem

Introduced in 1848 by chess composer Max Bezzel as the *eight queens puzzle*, it was later solved and extended by Franz Nauck to the *n queens problem*. It is a very well known constraint satisfaction problem, and consists in setting N

chess queens in a gameboard of $N \times N$ dimensions such as no pair of queens are attacking themselves [1]. In chess, queens can attack other pieces if they are in the same row, column or diagonal as them.

Some traditional forms of solving the problem include:

- *Recursive* algorithms: in each iteration they place a single queen in the board and change the problem from an n queens problem to a $n-1$ queens problem.
- A *brute-force* search algorithm: it considers all combinations of placing the queens in the board and then filters out the ones that put more than one queen in the same location or that put a pair of queens attacking themselves.
- A *backtracking* search algorithm with depth-first search: creates the search tree by taking into account one row of the board at a time, which eliminates most non-solutions very early in the execution.
- *Greedy* algorithms: even though they do not always find the optimal solution they can solve problems with greater size than the ones backtracking is able to solve.

3 A* algorithm

The A* algorithm [1,2] is an informed search algorithm. Its also an specialization of the BF (Best First) algorithm with a different evaluation function.

This algorithm evaluates nodes by adding $g(n)$, which is the best cost at that moment for going from the initial node to n, and $h(n)$, the estimated cost of reaching the goal from n.

The A* evaluation function can then be defined by:

$$f(n) = g(n) + h(n)$$

This means that $f(n)$ is the cost of the cheapest solution going through n.

Now, for the formal properties of A*, we have:

- **Admissibility:** a search algorithm is said to be admissible if it always finds the optimal solution. This means that the heuristic function h must be a positive estimation of the value of $h^*(n)$ for all nodes, that is, $h(n) \leq h^*(n) \forall n$.
- **Dominance:** if h_2 is better informed than h_1, then every node expanded by $A^*(h_2)$ is expanded by $A^*(h_1)$. This means that $A^*(h_2)$ dominates $A^*(h_1)$. For an heuristic function h_2 to be better informed than another one h_1 the following must be true: $h_1(n) < h_2(n) \leq h^*(n)$ for every n non final.
- **Monotony:** an heuristic function $h1$ is monotonous if for every pair of nodes n_1 and n_2 it is true that $h(n_1) \leq h(n_2) + c(n_1, n_2)$ being c the cost of going from n_1 to n_2. In graph-search, A* is said to be optimal if the $h(n)$ is

consistent (or monotonous). The consequences of A* being optimal are two. First, it is proven that the values of $f(n)$ along any path are nondecreasing. Second, if A* chooses a node n for expansion, we can conclude that the optimal path to n has already been found.

4 Genetic algorithms

Genetic algorithms [3,4,5] are metaheuristic search and optimization algorithms.

They were invented by John Holland at the University of Michigan in the 1970s and belong to a class of evolutionary algorithms based on the model of natural biological evolution depicted by Charles Darwin in his book *On the Origin of Species* [6].

Genetic algorithms start with a set of k randomly generated states, called the population. They perform the evolution of that population, selecting the fittest individuals to become the parents of the next generation and generating children from those parents until there are enough of them. This reproduction process is done using crossover and mutation operators.

A genetic algorithm has the following components:

- A **codification scheme**. The genetic algorithm needs some kind of codification in order to represent potential solutions. Each state is represented by a string over a finite alphabet. It can be a chain of binary digits, permutations, vectors, etc.
- A **fitness function**. This function is used to evaluate each individual and assign to them a value used for their selection as potential parents for the next generation. The fitness value is the non-negative value representing the performance of a given individual. The fitness function should return higher values for individuals with better performance.
- A way of **generating the initial population**. This method of creating initial populations can be done by the use of an heuristic function or, in the simplest case, making a random population disregarding their level of quality.
- A set of **genetic operators**.
  - **Selection:** operator that chooses a set of individuals from the current generation to be the parents for the next generation. Some selection operators include *Fitness-Proportionale Selection (or Roulette Selection)*, *Stochastic Universal Sampling* and *Tournament Selection*.
  - **Crossover:** operator that combines two or more parents to obtain offsprings that most likely will inherit characteristics from their parents. There are three classic ways of doing crossovers in vectors representing states: *One-Point*, *Two-Point* and *Uniform Crossover*.

    - **Mutation:** operator that makes minor random changes in the genetic structure of a chromosome in order to obtain a new range of genetic material. One example of a mutation operator is *bit-flip mutation*.

– Some **parameters**. Like the population size, number of generations, crossover probability, mutation probability, etc.

5 Application of A* algorithm to the N-queens problem

Two ways of modeling the search space have been defined for this problem:

– **Incremental:** in this version, the initial state is the empty gameboard and in each iteration a new queen is placed in a position such as it is not attacking or being attacked by any other queen in the board. This keeps going until the N queens are placed or until there are no locations in the board where the queen can be placed without breaking the constraints.

– **Complete:** in this version, the initial state is a board where the N queens are already placed. However, the queens can be in locations where they are attacking one another. In each iteration a queen is moved to a different position until a state where no pair of queens are attacking themselves is reached. There are three variants:

    - **Queens in first row:** in this variant the n queens are placed in the first row of the board.

    - **Queens in every column:** in this variant the n queens are placed in random positions of the board, so different executions can lead to different results.

    - **Queens in every column and row:** in this variant the n queens are again placed in random positions of the board, such that there are no pair of queens in the same row or column.

Different heuristics were implemented for solving the problem, but the following three were selected for the research:

– **Null heuristic:** heuristic function that always return zero. It is an admissible and a monotonous heuristic.

– **Number of attacking pairs:** heuristic function that estimates de number of necessary movements in order to achieve a solution by means of analysing the number of attacking pairs in a board where there are already N queens placed. It is not admissible, given that there can be situations in which the number of attacking pairs is greater than the cost of reaching the solution. For example, if a queen is attacked by two or more queens, and after moving it we reach the solution, that would mean that $h(n) > h^*(n)$, so the heuristic cannot be admissible.

– **Probabilistic estimation of solution:** heuristic function that estimates the probability of finding a solution from any given state. It is calculated by $h(n) = (N - k)/P(n)$, where N is the total number of queens to be placed, k the number of queens already placed, and $P(n)$ a probability of finding a solution from n. It is an admissible heuristic.

6 Applications of genetic algorithms to the N-queens problem

The genetic algorithm implemented for solving the N-queens problem has the following characteristics:

– **Codification scheme:** a chain of permutations that contains the rows in which the queens are located.

– **Fitness function:** it evaluates the number of pairs of queens not being attacked. It also implements fitness scaling, that is, to further differenciate the good chromosomes from the bad ones. The way it is done is by substracting from the fitness value of all individuals the fitness value of the worst in that generation.

– **Generating the initial population:** a chromosome is a random permutation of the numbers ranged from one to the size of the board such as no number is repeated.

– A set of **genetic operators**.

  • **Selection:** introduces elitism, that is, the best individual of that generation is carried over the next generation.

  • **Crossover:** implements the OX (Order crossover) operator. This method of crossover implies that the offspring will inherit the order and position of some of the genes of a parent and the relative order of the remaining genes of the other parent.

  • **Mutation:** swaps the content of two random positions of the permutation.

– **Parameters:**

  • Population size = 50

  • Mutation probability = 0.15

  • Variable number of generations according to N.

7 Experimental Research

The behaviours of the A* and genetic algorithm previously defined are going to be measured next. For this, a modified version of the *aima-java* project [7]

is used. As the name implies, these algorithms are implemented in the *Java* programming language.

For the enviroment used to test the algorithms the following information is listed:

- **Operating system:** Manjaro Linux *x86-64*
- **Kernel:** 5.10.30-1-MANJARO
- **Host:** 81DE Lenovo ideapad 330-15IKB
- **CPU:** Intel i7-8550U (8) @ 4.000GHz
- **RAM:** 8GB

This experimental research consists of a description of the dataset, a list of the results obtained for both algorithms and a comparison between their behaviours for solving the N-queens problem.

7.1 Dataset

The problems to be resolved are the following:

- **A*:**
  - **Complete version:**
    - **In the first row:** executed one time for each heuristic function and n in range 4, 6, 8, 16 and 32. The following heuristic functions will be tested:
      - *Null heuristic*
      - *Number of attacking pairs*
      - *Probabilistic estimation of solution*
    - **In every column:** executed ten times for each heuristic function and n in range 4, 6, 8, 16 and 32. The average and best execution will be listed. The following heuristic functions will be tested:
      - *Null heuristic*
      - *Number of attacking pairs*
      - *Probabilistic estimation of solution*
    - **In every column and row:** executed ten times for each heuristic function and n in range 4, 6, 8, 16 and 32. The average and best execution will be listed. The following heuristic functions will be tested:
      - *Null heuristic*
      - *Number of attacking pairs*

· *Probabilistic estimation of solution*

- **Incremental version:** executed one time for each heuristic function and n in range 4, 6, 8, 16 and 32. The following heuristic functions will be tested:
  * *Null heuristic*
  * *Number of attacking pairs*
  * *Probabilistic estimation of solution*

– **Genetic algorithm:** executed ten times for each n in range 4, 6, 8, 16 and 32. The average and best execution will be listed.

7.2 A* Results

The results obtained in the executions of A* are the following:

Null heuristic For the *null* heuristic the measurements obtained are listed below in the form of two tables. **Table 1** shows the expanded nodes for every version as well as the best, average and standard deviation for the completed version with queens in every column. Note that for $n = 8$ the average expanded nodes for the complete version with queens in every column is lower than for $n = 6$. This is because only two of ten executions were able to finish, and those were the ones with the lower expanded nodes. **Table 2** shows the same information but for the path cost of the solution.

Table 1. Expanded nodes for the *null* heuristic function

Expanded

N	Incremental	CompleteFirstRow	ComplEveryCol Best	ComplEveryCol Average	ComplEveryCol Std. Devi
4	15	70	6	68	58,256
6	149	15259	1667	8873,2	5215,761
8	1965	Not enough resources	1738	7276	7831,915

Table 2. Path cost for the *null* heuristic function

Path cost

N	Incremental	CompetelFirstRow	ComplEveryCol Best	ComplEveryCol Average	ComplEveryCol Std. Devia
4	4	3	1	2,2	0,919
6	6	5	3	4,2	0,632
8	8	Not enough resources	3	3	0,000

Number of attacking pairs For the *number of attacking pairs* heuristic the results obtained are listed below in the form of two tables. **Table 3** shows the expanded nodes for every version as well as the best, average and standard deviation for the completed version with queens in every column. **Table 4** shows the same information but for the path cost of the solution.

Table 3. Expanded nodes for the *number of attacking pairs* heuristic function

Expanded

N	Incremental	CompleteFirstRow	ComplEveryCol Best	ComplEveryCol Average	ComplEveryCol St
4	15	4	2	6,7	4,855
6	149	76	3	30,1	17,527
8	1965	14	11	37,2	24,521
16	Not enough resources	77	9	194	172,059

Table 4. Path cost for the *number of attacking pairs* heuristic function

Path cost

N	Incremental	CompleteFirstRow	ComplEveryCol Best	ComplEveryCol Average	ComplEveryCol Sto
4	4	3	2	2,8	0,632
6	6	6	3	4,1	0,738
8	8	8	4	4,9	0,876
16	Not enough resources	17	7	9	1,333

Probabilistic estimation of solution For the *probabilist estimation of solution* heuristic the results obtained are listed below in the form of two tables. **Table 5** shows the expanded nodes for every version as well as the best, average and standard deviation for the completed version with queens in every column. Note that for $n = 8$ the average and best case of the complete version with queens in every column are the same. This is because only one of ten executions was able to finish. **Table 6** shows the same information but for the path cost of the solution.

Table 5. Expanded nodes for the *probabilistic estimation of solution* heuristic function

Expanded

N	Incremental	CompleteFirstRow	ComplEveryCol Best	ComplEveryCol Average	ComplEveryCol
4	15	70	26	84,2	49,161
6	149	15259	576	8464,5	7253,468
8	1965	Not enough resources	20081	20081	0,000
16	Not enough resources	Not enough resources	Not enough resources	Not enough resources	Not enough reso

Table 6. Path cost for the *probabilistic estimation of solution* heuristic function

Path cost					
N	Incremental	CompleteFirstRow	ComplEveryCol Best	ComplEveryCol Average	ComplEveryCol
4	4	3	2	2,5	0,527
6	6	5	3	3,9	0,738
8	8	Not enough resources	3	3	0,000
16	Not enough resources	Not enough resources	Not enough resources	Not enough resources	Not enough reso

7.3 GA Results

Now the data obtained from the executions of the genetic algorithm will be shown in the form of two tables. **Table 7** shows the parameter values for the genetic algorithm. **Table 8** shows the average and best fitness values, the standard deviation and the number of solutions for each ten executions of the problem.

pop size	50
mutation prob	0.15
n generations	100

Table 7. Parameters of the genetic algorithm

N	Best Fitness	Average Fitness	Std. Deviation	Valid Solutions
4	6	6	0	10
6	15	15	0	10
8	28	28	0	10
16	120	119,5	0,527	5
32	493	492,3	0,949	0

Table 8. Data obtained for the genetic algorithm

7.4 Comparison of A* and GA results

Once both algorithms have been tested, an analysis of the data will be performed.

*A** In the case of A* a study on the best **space of search** and **heuristic function** was carried out. Of the three spaces of search present in this paper, these being both versions of the complete space of search and the incremental space of search, a better performance overall is observed in the case of the complete version with the n queens in **every column** of the board. In the case of heuristic functions, the one with the greatest results is the **number of attacking pairs** heuristic function.

GA In the case of the genetic algorithm a study focusing on the results given some fixed parameters was conducted. The algorithm finished the execution in every instance of n. However, as the value of n increased the number of found valid solutions started to drop in a quick manner, not being able to find a soluton for $n = 32$.

Comparison According to the data it can be said that the genetic algorithm was able to finish the execution of the problem for all cases studied whereas the A* could not in most spaces of search with some heuristic functions, probably due to the specs of the environment machine used for the tests. However, A* with the *number of attacking pairs* heuristic and the complete space of search placing the n queens in every column is able to find the solution for $n = 16$ in a more reliable way than the genetic algorithm. To sum up, genetic algorithms are more efficient but when n increases the proability of finding a solution lowers whereas A* with the right configuration can obtain great results but requires a greater computational process.

8 Conclusions

After performing both a theoretical and a experimental study of the problem and the algorithms, it can be concluded that there is no clear better solution for the N-queens problem. The A* algorithm always gives the optimal solution but requires big computational calculations whereas the genetic algorithm is more efficient but does not always give optimal solutions. With this conclusion the research is over.

References

1. J. Palma, R. Marín (Eds): Inteligencia Artificial. Técnicas, métodos y aplicaciones. McGraw Hill. 2008.
2. Stuart Russell, Peter Norvig: Artificial Intelligence: A Modern Approach. 3rd Ed. Pearson. 2010. Global Edition pp. 93-99.
3. Stuart Russell, Peter Norvig: Artificial Intelligence: A Modern Approach. 3rd Ed. Pearson. 2010. Global Edition pp. 126-129.
4. Sean Luke: Essentials of Metaheuristics. 2nd Ed. Lulu. 2013. pp. 36-45. Available at http://cs.gmu.edu/ sean/book/metaheuristics/.
5. MA. González, R. Varela: Tutorial on Genetic Algorithms. Computing department. University of Oviedo.
6. C. Darwin: On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life. John Murray. 1859.
7. Github's repository page for the *aima-java* project, https://github.com/aimacode/aima-java. Last accesed 1 November 2020.