University of California, Irvine

Introduction to Artificial Intelligence

CompSci 271, Fall 2017

N-queens Solver

Author:
Cheng Cai (ID: 95164901)

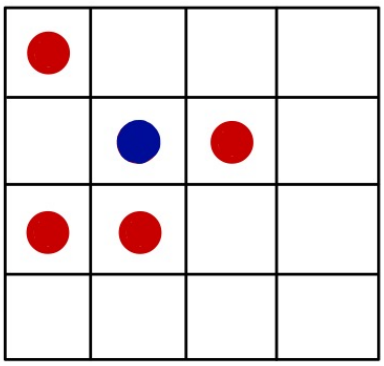
November 27, 2017

1 Introduction

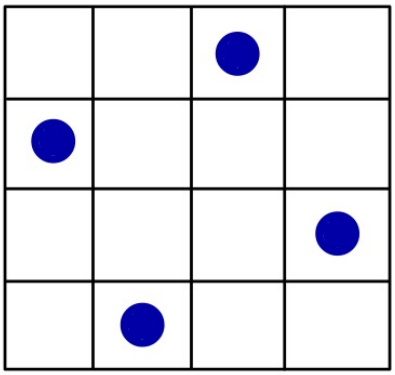
1.1 N-queens problem

N-queens is a classical artificial intellignece (AI) problem, and it is often used as a benchmark to test AI search problem-solving stategies [10]. It is still an NP hard problem since there is no known algorithms can find solutions in polynomial steps. Although some algorithms claim that, in some settings, they can find a solution of N-queens in polynomial steps [10] [7], these algorithms are not guarenteed to provide an answer. Whether they can provide an answer depends on the initial setting of the permutation, however the initial setting is random choosen [10].

Here I briefly describe the N-queens problem. The goal is to place n queens onto the board without letting them attack with each other. Given the input number n, we can know the size of the board, since the board has n rows and n columns. Therefore, there are n^2 positions on the board, on which the n queens can be placed. One queen will attack others if there is another queen on its row, column or diagnols. Here is a concrete example: when $n = 4$, the board consists 16 posisions. One possible attacking example is shown in Figure 1a and one of the solutions is shon in Figure 1b. In other words, the goal is to find all the valid **configurations** (one configuration is a startegy to place n queens on the board.)



(a) An attacking example of 4-queens problem: the blue queen is attacking all the red queens around it.



(b) A solution of 4-queens problem

Figure 1: 4-queens problem

Table 1: Number of solutions of N-queens puzzle.

n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**	**25**	**26**	**27**
fundamental	1	0	0	1	2	1	6	...	275,986,683,743,434	2,789,712,466,510,289	29,363,791,967,678,199
all	1	0	0	2	10	4	40	...	2,207,893,435,808,352	22,317,699,616,364,044	234,907,967,154,122,528

The most famous example of N-queens problem is the eight queens puzzle, since it was first published back to early 1900s [6]. All of its solutions can be found on Wikipedia [4]. In next section, I will talk about some existing solutions of the N-queens problem.

1.2 Existing results

There are two options of finding the solutions of N-queens problem: find one solution and find all the solutions. The first one will terminate the algorithm as long as it find one valid solution and report it, but the second one will continue search untill it find all the solutions for that problem and report all the solutions as well as the number of the solutions. It is easy to imagine that, the first one is easier than the second option. Some works only focus on finding just one solution, and they can solve the problem with a very large n (e.g. $500,000$) [10]. However, the reason for these algorithms of finding just one solution is that they always use some more efficient ways to get the answer with much less time, and these efficient ways are not always guarenteed to give an answer at the end (not sound) [10]. If they use a method that is garenteed to provide a solution, they can just run the algorithms multiple times to get all the solutions.

In this report, my goal is to find the existing most efficient (in terms of time performance) algorighm of solving the N-queens puzzle. The algorithm should report all the valid solutions along with the number of them. The most recent existing result of N-queens puzzle only reports the solutions up to $n = 27$ (see Table 1. The source of data is from the "The On-Line Encyclopedia of Integer Sequences (OEIS)" [2] [1]. Fudamental solutions are the ones after filtering the mirrored ones of all the solutions.)

2 Solutions

2.1 Naive algorithm

The naive way of solving N-queens puzzle is to enumerate all the possible configurations to find the right ones. This indeed is time as well as memory consuming when n increase. One simple way of optimization is, instead of generating all the configurations, only gernerating the configurations that for each column or each row there is only one queen. Still the time of finishing the whole process is going to be unacceptable.

2.2 Backtracking with CSP

In stead of generating all the configurations and then iterate over them to find the valid answers, we put a new queen onto the current board and check whether it will attack the existing queens that are already on the board. If it is yes, then we take that queen off and try another position. If it is no, we leave the queen in that position and pick another queen that is not on board currently as the new queen and repeat the process. Another thing we need to do is to remember all the actions we did, and if we found that there is no positions allow a new queen to be placed in, then it is time to reverse the previous process, i.e. take off the last placed queen before the current one, and then try to place it somewhere else. This process essentially is doing the depth first search on a tree. Each node on the tree is a configuration and the edges are the actions taken on the parent node and the child nodes are the result. The search process is called backtracking [9], since it will search down and if it find a unrealized path it will backtrack to the parent nodes. Figure 2 is an example of backtracking on the 4-queens puzzle.

In addition, instead of looking at every queen in the configuration and do the computation every time, we need to build some data structures to help us to check the attacking condition by simply scanning the data structures. This is called constraint guided search [9]. Everytime we take an action, it must statisfy the constraint, and here the constraint is no queens stay on the same columns, rows, or diagnols. The constraint satisfication problem is shorted for CSP. In my implementation, there is an array of bits (each element's value is 0 or 1) to record, for every column, row and diagnol, whether there is a queen already been placed. The array's size is determined by the number n: the number of columns and rows are both n, and the number of two diaganols

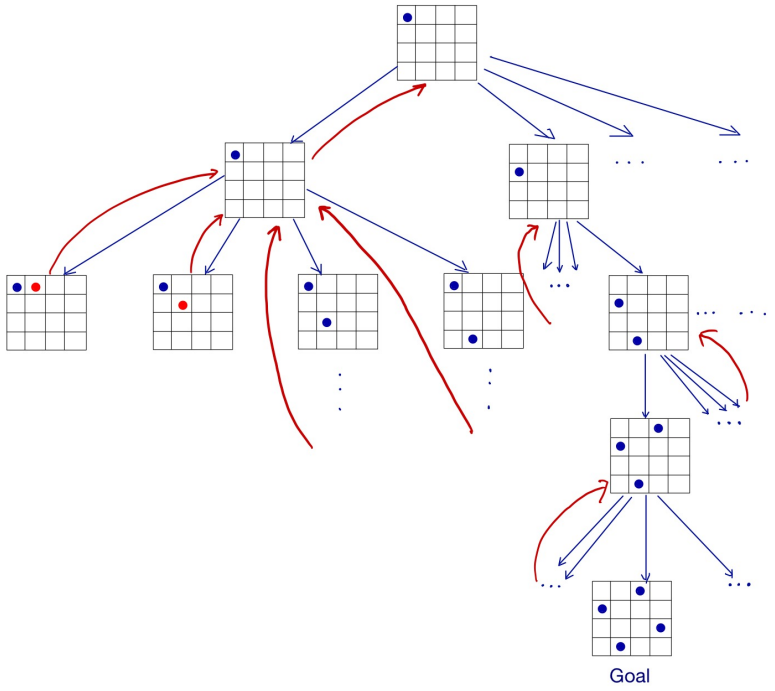


Figure 2: An example of backtracking process (incomplete) for 4-queens puzzle. Each node is a configuration. The red lines indicate the backtrack actions.

are both $2n-1$, so the total number is $4n-2$.

In my implementation, there is a global array for the CSP, and it does the search rescursively. Before querying a child node, it will remember the action that is going to be taken, and after the querying is done it will reverse the actions that were taken. More details of the implementaion will be dicussed in the evaluation section.

2.3 Algorithm X

Backtracking is the good way of searching process, however because the representation of each node that the backtracking works on is not good enough, the depth as well as the number of branching of the search tree is very large. This will make the searching time longer, because there are too many nodes will be traversed.

In this section, we will discuss another techquen that can reform the N-

queens problem into **exact cover problem** [5] which can make the depth and branching number much smaller. Combining a very efficient search algorithm called **Alogrithm X** in which another techque called **Dancing Links alorithm (DLX)** [3] [8] is used, the search tree is going to be much smaller. Here I am going to expain what the exact cover problem is, how we convert the N-queens problem to exact cover problem, and how the dancing links works in Algorithm X. Therefore, we can first convert our problem into Exact Cover problem and use Algorithm X to solve it fast.

2.3.1 Exact cover problem

Exact cover problem is defined like this: given a set X, and a collection S of X's subsets, the goal is to find another collection S^* in which every subset y is in S and dose not intersect with other subset, as well as the union of all the subset in S^* is exactly set X. Here we use an example to explain it. Suppose $X = 1, 2, 3, 4, 5, 6, S = A, B, C, D, E, A = 1, 3, B = 2, 4, 5, C = 1, 6, D = 3, E = 1, 3, 6$, then the S^* should be $S^* = B, E$. We can use the a matrix to represent this example (see Figure 3.)

	1	2	3	4	5	6
A	1	0	1	0	0	0
B	0	1	0	1	1	0
C	1	0	0	0	0	1
D	0	0	1	0	0	0
E	1	0	1	0	0	1

(a) The matrix representation of the example of exact cover problem.

	1	2	3	4	5	6
A	1	0	1	0	0	0
B	0	1	0	1	1	0
C	1	0	0	0	0	1
D	0	0	1	0	0	0
E	1	0	1	0	0	1

(b) A solution of the example. Every column has only one red 1s when we choose B and E as the solution.

Figure 3: An example of exact cover problem

2.3.2 Conver N-queens to exact cover problem

If we think about the array we mentioned in Section 2.2, we can notice that each element of the array can only have one 1. That is exactly the columns mentioned before. Each row in the matrix is a position on the

board. Therefore, there are n^2 rows. Thus, we can initialize the matrix for our exact cover problem.

2.3.3 Dancing links

This technque can make operation on the matrix cheap.

2.4 DLX with heuristic

2.5 Other algorithms

3 Evaluation

4 Conclusion

5 Future work

6 Reference

References

[1] A000170 - OEIS.

[2] A002562 - OEIS.

[3] Knuth's Algorithm X, Sept. 2016. Page Version ID: 738699208.

[4] Eight queens puzzle, Nov. 2017. Page Version ID: 811616720.

[5] Exact cover, Aug. 2017. Page Version ID: 796864531.

[6] Ball, W. W. R. *Mathematical recreations and essays*. MacMillan, 1914.

[7] Bernhardsson, B. Explicit Solutions to the N-queens Problem for All N. *SIGART Bull. 2*, 2 (Feb. 1991), 7–.

[8] Knuth, D. E. Dancing links. *arXiv:cs/0011047* (Nov. 2000). arXiv: cs/0011047.

[9] Russell, S., Norvig, P., and Intelligence, A. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs 25* (1995), 27.

[10] Sosic, R., and Gu, J. A Polynomial Time Algorithm for the N-Queens Problem. *SIGART Bull. 1*, 3 (Oct. 1990), 7–11.