COMP417 Artificial Intelligence
Term Project Report - Draft
Local and Constraint Satisfaction Search Algorithms for N-Queens Problem

Pantourakis Michail AM 2015030185
Proestakis Ioannis AM 2009030018
School of Electrical and Computer Engineering
Technical University of Crete

17 May 2018

1 Introduction

The n-queens problem, a combinatorial search problem, concerns the non-attacking (horizontally, vertically, and diagonally) placement of n queens on a $n \times n$ chessboard. The n-queens and similar Constraint Satisfaction Problems (CSPs) are classical examples of the limitations of simple backtracking search, with an exponential worst case time complexity that renders solving for large n impractical [1, 2]. Although many efficient heuristics have already been proposed for this problem [3–7], it still is a popular test bed for new Artificial Intelligence (AI) search problem methods. Whilst a toy problem per se, it has found some practical applications such as VLSI routing and testing, data compression, maximum full range communication and parallel optical computing [3, 4].

This problem has (at least) two variants depending on the desired number of solutions. A single solution can actually be found trivially without search, since explicit solutions exist $\forall\ n \geq 4$ [8]. On the other hand, finding all possible solutions is non-trivial. In this project we will focus on the former variant, implementing and comparing the performance of two different search algorithms: a local search algorithm, and a constraint satisfaction method. We will first describe the problem mathematically and define the performance indicators for our comparison. In Section 3 we will describe the implemented algorithms, and we will subsequently illustrate and discuss their performance.

2 Problem Formulation

2.1 Mathematical Model

A naive formulation would allow any arrangement on the chessboard, resulting in a huge number of combinations (e.g. 3×10^{14} for $n = 8$). To reduce the state space, we distribute them so that each column contains only one queen. This restriction ensures that there will be no vertical conflicts, resulting in just 2057 candidate solutions [5].

Therefore, let array $Q = (q_1, ..., q_n : q_i \in \{1, ..., n\}\ \forall i \in \{1, ..., n\})$ be an n-queens placement, where i is the column number, and q_i is the row number of the i-th queen. To ensure that we have no horizontal or diagonal collisions, we need two more constraint equations. The first constraint is expressed as $q_i \neq q_j,\ \forall i \neq j$, namely no pairs of columns should be in the same row. The second constraint can be expressed as $|i - j| \neq |q_i - q_j|,\ \forall i \neq j$, namely no pair can be in the same diagonal.

Having defined the constraint and state representation formulas, the objective function that returns the total amount of direct and indirect collisions of Q is given by:

$$n_c(Q) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c(q_i, q_j) \tag{1}$$

$$c(q_i, q_j) = \begin{cases} 1, & q_i = q_j \\ 1, & |i-j| = |q_i - q_j| \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

2.2 Performance Indicators

In order to assess the quality of our methods, we will employ up to five (exact number pending) different performance indicators: 1) average time duration, 2) average memory used, 3) average number of $n_c(Q)$ evaluations, 4) average number of $c(q_i, q_j)$ evaluations, and 5) average n_c (if sub-optimal solutions are allowed, otherwise this metric will always equal zero). Time and memory measurements will give an estimate of the time and space complexity of each method respectively. However, absolute time is hardware-dependent, and may even vary in the same machine due to other operating system processes. Hence, the average number of $c(q_i, q_j)$ calls shall offer a hardware and software-independent time metric, where we could assume that each call costs $O(1)$ time, whereas the average number of $n_c(Q)$ shall provide a counting-independent time complexity indicator.

3 Methods

We selected two algorithms to implement in MATLAB: 1) the min-conflict algorithm as a representative local search method, and 2) forward checking with minimum remaining values (FC-MRV) algorithm as a CSP solver [5]. We chose these two algorithms since previous results have illustrated their good performance, especially in comparison to simpler backtracking-based methods [5].

3.1 Min-Conflicts Algorithm

Algorithm 1 MIN-CONFLICTS(n, maxSteps)

```
1: randomize an initial solution
2: steps = 0
3: while $n_c \geq 1$ AND steps < maxSteps do
4:     choose a random column i
5:     if current row at column i has conflicts then
6:         Calculate number of conflicts for each row
7:         Choose newRow with lowest number of conflicts
8:         solution[i] = newRow
9:     end if
10:    steps++
11: end while
12: return solution
```

3.2 FC-MRV Algorithm

Algorithm 2 FC-MRV-MAIN(n)

```
1: Initialize empty solution
2: Initialize currDomain = $\mathbf{1}^{n \times n}$
3: for all i : rows do
4:     solution[1] = i
5:     newSolution = FC-MRV(currDomain, solution, i, 1)
6:     if newSolution not empty then return newSolution
7:     end if
8: end for
9: return empty solution
```

Algorithm 3 FC-MRV(currDomain, solution, row, column)

```
1: for all i : unassigned columns do
2:     UpdateDomain(currDomain, row, column, i)
3:     if no available rows left for column i then return empty solution
4:     end if
5: end for
6: if all columns have been assigned with a row then return solution
7: end if
8: newColumn = MRV(currDomain, solution)
9: for all newRow : available rows for newColumn do
10:     solution[newColumn] = newRow
11:     newSolution = FC-MRV(currDomain, solution, newRow, newColumn)
12:     if newSolution not empty then return newSolution
13:     end if
14: end for
15: return empty solution
```

Algorithm 4 UpdateDomain(currDomain, q_i, i, j)

```
1: for all $q_j$ : rows do
2:     if $c(q_i, q_j) == 1$ then currDomain[$i$,$j$] = 0
3:     end if
4: end for
```

References

[1] R. Sosic and J. Gu. “A Polynomial Time Algorithm for the N-Queens Problem”. In: *SIGART Bull.* 1.3 (Oct. 1990), pp. 7–11. ISSN: 0163-5719.

[2] H. S. Stone and J. M. Stone. “Efficient search techniques - an empirical study of the N-Queens problem”. In: *IBM Journal of Research and Development* 31.4 (1987), pp. 464–474.

[3] R. Sosic and J. Gu. “Fast search algorithms for the n-queens problem”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 21.6 (1991), pp. 1572–1576.

[4] X. Hu, R. C. Eberhart, and Y. Shi. “Swarm intelligence for permutation optimization: a case study of n-queens problem”. In: *Swarm intelligence symposium, 2003. SIS’03. Proceedings of the 2003 IEEE.* IEEE. 2003, pp. 243–246.

[5] S. J. Russell, P. Norvig, et al. *Artificial intelligence.* Prentice Hall/Pearson Education, 2003.

[6] M. R. Engelhardt. “A group-based search for solutions of the n-queens problem”. In: *Discrete Mathematics* 307.21 (2007), pp. 2535–2551. ISSN: 0012-365X.

[7] K. Agarwal, A. Sinha, and M. H. Bindu. “A novel hybrid approach to N-queen problem”. In: *Advances in Computer Science, Engineering & Applications.* Springer, 2012, pp. 519–527.

[8] B. Bernhardsson. “Explicit solutions to the N-queens problem for all N”. In: *ACM SIGART Bulletin* 2.2 (1991), p. 7.