CISC 481 Programming Assignment 2

October 18, 2020

Assignment Objectives

After completing this assignment, you'll understand the inner workings of the following search algorithms:

- AC-3 and Maintaining Arc Consistency
- Backtracking Search for CSPs

You'll also gain some experience setting up a basic web service to visualize the results of your backtracking search.

The n-Queens Problem

Another classic problem studied in Artificial Intelligence, the goal is to place queens on a chess board such that no pair of queens threaten one another. The standard instance is the 8-queens problem, where you need to place 8 queens on a standard (8×8) chess board. In this assignment, you'll want your code to be able to solve up to 32-queens (that is, place 32 queens on a 32×32 board).

Representing the Problem

[My examples here are in Lisp, but feel free to use something similar that's easy to parse in your target language.]

Recall that a Constraint Satisfaction Problem Consists of a set of *variables*, a set of *domains* (one for each variable), and a set of *constraints* over the variables.

One way to represent an n-queens problem as a constraint satisfaction problem is to have a variable for each queen, and the domain for each variable be the set of integers 1 thru n. Assigning a value i to variable q_j says that queen q_j will be placed in row i. Figure 1 illustrates how we might structure the 4-queen problem.

Part 1 [10 pts]

Write a function `revise` that takes a CSP such as the one in Figure 1 and the names of two variables as input and modifies the CSP, removing any value in the first variable's domain where there isn't a corresponding value in the other variable's domain that satisfies the constraint

```
(:VARIABLES ((:Q1 1 2 3 4) (:Q2 1 2 3 4) (:Q3 1 2 3 4) (:Q4 1 2 3 4))
 :CONSTRAINTS
 (((:Q1 :Q2) (1 3) (1 4) (2 4) (3 1) (4 1) (4 2))
  ((:Q1 :Q3) (1 2) (1 4) (2 1) (2 3) (3 2) (3 4) (4 1) (4 3))
  ((:Q1 :Q4) (1 2) (1 3) (2 1) (2 3) (2 4) (3 1) (3 2) (3 4) (4 2) (4 3))
  ((:Q2 :Q3) (1 3) (1 4) (2 4) (3 1) (4 1) (4 2))
  ((:Q2 :Q4) (1 2) (1 4) (2 1) (2 3) (3 2) (3 4) (4 1) (4 3))
  ((:Q3 :Q4) (1 3) (1 4) (2 4) (3 1) (4 1) (4 2))))
```

Figure 1: A Lisp representation of the 4-queens problem as a CSP. **NB** that we've combined the variables and their domains into association lists, one for each variable. The constraints are lists of pairs, the first pair being the two variables involved in the constraint, and the remaining pairs comprising a complete list of compatible assignments for those two variables.

between the variables. The function should return a boolean indicating whether or not any values were removed.

Part 2 [20 pts]

Implement AC-3 as a function which takes as input a CSP and modifies it such that any inconsistent values across all domains are removed. The function should return a boolean indicating whether or not all variables have at least on value left in their domains.

Part 3 [5 pts]

Write a function `minimum-remaining-values` that takes a CSP and a set of variable assignments as input, and returns the variable with the fewest values in its domain among the unassigned variables in the CSP.

Part 4 [35 pts]

Implement a *backtracking search* which takes a CSP and finds a valid assignment for all the variables in the CSP, if one exists. It should leverage your AC-3 implementation to maintain arc consistency.[1] When Choosing a variable to assign, it should use your minimum remaining values heuristic implementation. Along with the solution to the CSP, your search should return the order in which it assigned the variables, and the domains of the remaining unassigned variables after each assignment.

Part 5 [30 pts]

Now you'll implement a web-based visualization of your backtracking search.[2] It should allow the user to specify a problem size of 8, 12, or 16 queens. It should also allow the user to force the

[1] Technically, the MAC algorithm initializes the queue to only the edges involving the variable that's just been chosen for assignment - but doing a full run of AC-3 will achieve the same thing, at the cost of maybe doing a little bit of unnecessary work.

[2] For Lisp, see Caveman 2 for a relatively easy to get up and running web framework.

location of any queen to be a particular row.[3] It should then show the board state at each step of the problem[4]. The board state should show which queens have already been placed at that step, as well as the restricted domains of the remaining queens. See Figure 2 for an example of how you might draw your board.

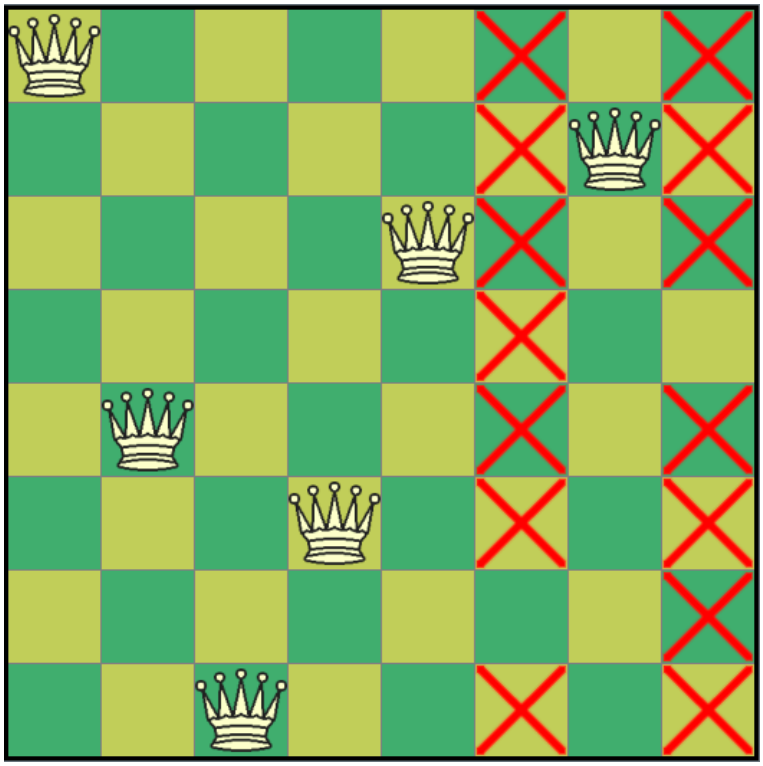

Figure 2: On this board, Queens 1, 2, 3, 4, 5, and 7 have already been placed. The crossed out squares in columns 6 and 8 correspond to those positions that are inconsistent with the current assignment. From this, we can see that Queens 7 and 8 each only have one possible placement from this state.

Submitting

You should submit all of your code - *document it appropriately, as you'll be graded on style.* You should also submit instructions for how to setup the environment for your web service and how to run it.[5,6]

[3] An easy way to implement this is to just restrict the domains of whatever queens the user fixes to be the singleton values they chose when generating the CSP to give to your search.

[4] For n queens there will be n steps.

[5] If you're using Lisp, you can just put any extra dependencies you added in the .asd file for the project.

[6] *e.g.* For Python this would include a requirements file that you would get from running `pip freeze`.