1 More $n$-Queens

If you came to last week’s section, you remember we were able to solve the $n$-queens problem with $n = 8$ in around 1 second, using depth first search. What about $n = 100$?

The first solution found for the 100-queens problem by our CSP solver in 0.21 seconds.

(1) Formulate the $n$-queens problem as a CSP:

   Variables:
   Each column of the board is a variable.

   Domains:
   Each variable can take a value in $[0, n]$

   Constraints:
   Explicit constraints list acceptable values for each pair of constrained variables. Implicit constraints are functional. See code for details.

(2) Draw the constraint graph for your formulation with $n = 4$.
   Each variable is a node; the graph is fully connected.

(3) Use backtracking search with forward checking to solve 4-queens. Refer to the python code on the next page. How many backtracks are required?
   4 backtracks, assuming 0-indexed variable and value ordering
def recursive_backtracking(assignment, csp):
    # base case
    if len(assignment) == len(csp.vars): return assignment

    # choose a variable
    var = select_unassigned_variable(assignment, csp)

    # try each possible value of that variable
    for val in order_domain_values(var, assignment, csp):
        # if no immediate conflicts, add this to the assignment
        if csp.num_conflicts(var, val, assignment) == 0:
            csp.assign(var, val, assignment)
            result = recursive_backtracking(assignment, csp)
            if result != None: return result

        # if this value doesn’t work, backtrack (unassign it and try the next value)
        csp.unassign(var, assignment)

    # tried everything and nothing worked
    return None

Python code for recursive backtracking search.

2 Improving Backtracking Search

Explain how to incorporate each of these additions into the backtracking search code above:

(1) Forward Checking
Assignment now includes forward checking; Must keep track of pruned var:val mappings; Unassignment includes replacing pruned var:val items; No longer need a check in backtracking search for conflicts—new assignments are guaranteed to be legal.

(2) Minimum Remaining Values (MRV) heuristic
Selecting the next unassigned variable now chooses the variable with the smallest remaining domain.

(3) Arc Consistency
Similar to forward checking. See code for details.

3 Sudoku Challenge

With forward checking and MRV, our CSP implementation can solve any standard 9x9 Sudoku puzzle quite quickly. But 16x16 Sudoku is more challenging. Can you modify, add to, or rewrite our code to solve the “reallyHardPuzzle” in under 1 second?