CS 486 — Lecture 4: Constrained Satisfaction Problems (CSP)

1 Introduction to CSPs

- What is the difference between a CSP to a normal search problem?
- A typical search algorithm is unaware of the internal structure of states — it's like if it treats each state as some black box, only knowing how to generate successors and testing whether the state is a goal!
- For example, the four queens example — a typical SA cannot understand that adding two queens beside each other is an instant failure.
- So, knowing the internal structure of the states can help design a smarter algorithm.
- Currently, the search algorithm can only understand "this is not the goal state, keep going" — we can improve it to recognize dead ends and allow for immediate backtracking.
- Each state contains:
  - A set $X = \{X_1, X_2, \ldots, X_n\}$ of variables.
  - A set D of domains.
  - A set C of constraints.
- A solution is an assignment of values to all variables that satisfy all the constraints.
- For example, for the 4Q problem:
  - Variables — we assume that exactly one queen is in each column; this is obvious as fitting two queens in a column would instantly fail. This also simplifies our problem.
  - Thus, define four variables, X_0, X_1, X_2, X_3. X_i represents the row position of the queen in column i, $i \in [0, 3]$.
  - Define our domain as $D_i = 0, 1, 2, 3$ for each X_i.
  - Define our constraints as no two queens can be in the same row or diagonal. Logically, we define it as:

$$\forall i \forall j (i \neq j) \rightarrow ((X_i \neq X_j) \wedge (|X_i - X_j| \neq |i - j|))$$

2 Solving a CSP

2.1 Backtracking Search

- Let us design a backtracking solution for the 4Q problem.
- We will assume we place queens from left to right. We also assume we ensure the constraints are satisfied.
- Our initial state is an empty board. Our goal state is 4 queens on the board with none attacking another.
- Our successor function is adding a queen to the leftmost empty column s.t. it is not attacked by another existing queen.
- This is similar to the typical Sudoku backtracking solution.
- This search tree is a form of DFS, but with a check to stop if we hit a dead end!

2.2 Arc-Consistency

- Suppose we started the 4Q problem with $x_0 = 0$. $x_2 = 1$ and $x_2 = 3$ do not lead to a solution, ever!
- We can show our constraints for the problem in a constraint network/graph.
  - We have circular nodes to represent variables.
  - We have (undirected) arcs connecting the variables to constraints; the constraints are represented by rectangular nodes.
  - In some graphs, we combine constraints together to be convenient.
  - We define the number of variables required for a constraint as the arity. For example, in the 4Q problem, we have only binary constraints.
  - We can typically just use binary constraints. Why not unary or higher arity constraints?
    - Unary just means "allow some things and disallow all that are not this thing" — so we can just "preprocess" and eliminate these values and thus, no need to mention the unary constraint!
    - Any constraint of arity > 2 can be decomposed to a binary constraint.
  - We define an arc $\langle X, c(X, Y)\rangle$ as **arc-consistent** iff $\forall v \in D_X \exists w \in D_Y : (v, w)$ satisfies the constraint $c(X, Y)$.
  - For example, given $D_X = \{1, 2\}, D_Y = \{1, 2, 3\}$, then X is arc-consistent with the constraints of $X < Y$, while for $D_X = \{1, 2\}$ and $D_Y = \{1, 2\}$, then X is not arc-consistent with the same constraints.
  - Note that arc-consistency is not symmetric.
- AC-3 Arc Consistency Algorithm:

```
def ac3():
    put every arc in the set S
    while S is not empty do:
        select and remove <X, c(X, Y)> from S
        remove every value in D_X
            that does not have a value in
            D_Y that satisfies the constraint c(X,Y)
        if D_X was reduced:
            if D_X is empty:
                return False
            for every Z not equal to Y:
                add <Z, c'(Z, X)> to S
    return True
```

  - An algorithm used for the solution of CSPs (arc consistency).
  - What happens when we remove a value from a domain in regards to arc-consistency? It's possible that by doing so, we have invalidated a formerly arc-consistent arc!
  - Does the order in which arcs are considered matter? Nope.
  - There are three possible outcomes of the AC-3 algorithm:
    1. If the domain of a variable is empty, no solution exists.
    2. If every domain has exactly one value left, then we found a unique solution.
    3. Every domain has at least one value left, and at least one domain has multiple values left. In this case, we don't know if there are multiple solutions, no solution, a unique solution, etc. As such, AC-3 fails to give a definite answer here.
  - Is AC-3 guaranteed to terminate? Yes.
  - The runtime of AC-3 is $O(cd^3)$. There are n variables, size of each domain is $\leq d$ with c binary constraints. The AC check takes $O(d^2)$.