Lecture 4, Jan 30, 2024

Constraint Satisfaction Problems (CSPs)

Definition

Constraint Satisfaction Problem: A *CSP* comprises of 3 components:

1. A set of variables $X = \{ x_1, x_2, \ldots, x_n \}$
2. A set of domains for each variable: D: $\{ D_{x_1}, D_{x_2}, \ldots, D_{x_3} \}$
3. A set of constraints C relating the variables

The problem is to find a value for each of the variables in its domain that satisfies all the constraints.

- Example: For the 4-queens problem:
  - Variables: $\{ x_1, x_2, x_3, x_4 \}$
  - Domain: for each variable: $D_{x_i} = \{ 1, 2, 3, 4 \}$
  - Constraint: NoAttack(x_i, x_j) (true if queen x_i can attack x_j)
    - We can express this in a table, giving the value of NoAttack for every combination of x_i, x_j
- In general CSPs are NP-hard – no polynomial time solution exists
- But we can use heuristics to do better in a lot of the problems that arise in real life

Backtracking Search

- Assign each of the variables some value, and then check if the constraints are satisfied
- If the constraints are not satisfied, revert the last variable that we assigned – this is the process of *backtracking*
- If no value of the last variable works, then we go back one more variable and pick another value for that one, and so on
- This is essentially a brute force search if we pick values for the variables sequentially
  - However we can also use heuristics to aid our search

Algorithm 1 BacktrackingSearch(*prob,assign*)

```
1. if AllVarsAssigned(prob,assign) then
2.   if IsConsistent(assign) then
3.     return assign
4.   else
5.     return failure
6. var←PickUnassignedVar(prob,assign)
7. for value∈ OrderDomainValue(var,prob,assign) do
8.    assign← assign ∪ (var = value)
9.    result ← BacktrackingSearch(prob,assign)
10.   if result != failure then return result
11.   assign ← assign \ (var=value)
12. return failure
```

Figure 1: Basic backtracking search algorithm.

- The pseudocode above is a template for the backtracking search algorithm
  - Each level of the recursion picks a variable to set, goes through all values of that variable and checks if any of them work
  - We can specify different implementations for PickUnassignedVariable and OrderDomain
- One simple improvement we can make is to only assign variables to values that satisfy all the constraints

- Otherwise we would do a lot of meaningless searches when we pick a value that violates a constraint and keeps assigning others
- e.g. for N-queens, check that the new queen cannot attack any previous queens before placing it

Algorithm 2 BacktrackingSearch(*prob*,*assign*)

```
1. if AllVarsAssigned(prob,assign) then
2. var←PickUnassignedVar(prob,assign)
3. for value∈ OrderDomainValue(var,prob,assign) do
4.     if ValIsConsistentWithAssignment(value,assign) then
5.         assign← assign ∪ (var = value)
6.         result ← BacktrackingSearch(prob,assign)
7.         if result != failure then return result
8.         assign ← assign \ (var=value)
9. return failure
```

Figure 2: Improved backtracking search algorithm.

- Can we do better and reduce our backtracks even more?
  - Every time we assign a variable, it reduces the domain that the other variables can take based on constraints
  - The domain reduction (restrictions) on the other variables can propagate to even more variables
  - Every time we do an assignment, we call the inference function, which restricts the domain further based on the constraints and the new variable value
    * This can look at multiple constraints at the same time but often we stick to just 1

Algorithm 3 BacktrackingSearch_with_Inference(*prob*,*assign*)

```
1. if AllVarsAssigned(prob,assign) then
2. var←PickUnassignedVar(prob,assign)
3. for value∈ OrderDomainValue(var,prob,assign) do
4.     if ValIsConsistentWithAssignment(value,assign) then
5.         assign← assign ∪ (var = value)
6.         inference← Infer(var,prob,assign)
7.         if inference != failure then
8.             assign← assign ∪ inference
9.             result ← BacktrackingSearch(prob,assign)
10.            if result != failure then return result
11.        assign ← assign \ {(var=value) ∪ inference
12. return Failure
```

Figure 3: Improved backtracking with inference.