CS3243: Introduction to Artificial Intelligence
Lecture Notes 7: Constraints - II

1. Recap of Backtracking Search

In the previous lecture, we have covered the algorithm **Backtracking Search**.

```
BacktrackSearch(prob, assign):
    if AllVarsAssigned(prob, assign):
        return assign
    var ← PickUnassignedVar(prob, assign)
    for val in OrderDomainValue(var, prob, assign):
        if (val is consistent with assign):
            assign.add(var=val)
            inference ← Infer(prob, var, assign)
            assign.add(inference)
            if (inference != failure):
                result ← BacktrackSearch(prob, assign)
                if (result != failure):
                    return result
        remove {var=val} and inference from assign
    return failure
```

Take note that we can also replace the recursive call of Backtrack Search with a Local Search instead.

However, we have not covered the manner in which the search algorithm should:

- Pick an unassigned variable from the set of variables (`PickUnassignedVar`)
- Pick a value from the domain of an unassigned variable (`OrderDomainValue`)
- Make an inference on the values which other variables cannot take (`Infer`)

2. Implementation of Inference

a. Mathematical Representation

The representation of an `inference` on a variable X is defined as $\{X \notin S\}$, where S is a set of values in which the variable X cannot take. For example, $X \neq 1$ is equivalent to $X \notin \{1\}$.

b. Computing the Domain of a Variable

We now define a function called `ComputeDomain`, defined as follows.

```
ComputeDomain(X, assign, inference)
    return the set of values of X that X can take under the current
assignment and inference
```

Example 1:

- $Domain(X) = \{1,2,3,4\}$
- $assign = [\{X = 1\}]$
- $inference = [\{(X \notin \{2\}), (X \notin \{3\})\}]$

→ **ComputeDomain**`(X, assign, inference)` will return $X = \{1\}$, because X has already been assigned a variable.

Example 2:

- $assign = [\{Y = 1\}]$
- $inference = [\{(X \notin \{2\}), (X \notin \{3\})\}]$

→ **ComputeDomain**`(X, assign, inference)` will return $X = \{1, 4\}$

c. Pseudocode for Inference

With that, the following is the pseudocode for the function `Infer`.

```
Infer(prob, var, assign)
    inference ← empty
    varQueue ← [var]
    while (varQueue is not empty):
        Y ← varQueue.pop()
        for each constraint C where Y ∈ Vars(C)
            for all X ∈ Vars(C)\Y:
                S ← ComputeDomain(X, assign, inference)
                for each v in S:
                    if (no valid value exists in Var(C)\X such
                    that c[X↦v] is satisfied):
                        inference.add(x ∉ {v})
                T ← ComputeDomain(X, assign, inference)
                if (T = {}):
                    return failure
                if (S ≠ T):
                    varQueue.add(X)
    return inference
```

Take note that X↦v refers to "X is substituted with v".

It looks like the process for inference is actually very slow. Alternative implementations for the above process include:

- **Choice 1 (Forward Checking):** Don't add anything to `varQueue`.
- **Choice 2:** Instead of "`if (S ≠ T)`", we replace it with "`if (|T| = 1)`"

3. Pick Unassigned Variable

The problem of the best order to pick the next unassigned variable in Backtracking Search is still being discussed in computer science conferences today. One of the approaches is the **Minimum Remaining Value (MRV)** heuristic, where we choose the variable with the fewest possible values.

4. Order Domain Value

When picking a value from a variable's domain, we can try to pick a value that is likely to succeed.

5. Constraint Satisfaction Problems (CSP)

Constraint Satisfaction Problems are actually NP-complete (i.e. their time complexity is non-deterministic polynomial time). The following are variants of the problem.

- **Binary CSP:** Every constraint is defined over two variables. It is also NP-Complete.
- **Boolean CSP:** Domain of every variable is {0, 1}. It is also NP-Complete.
- **2-SAT:** A combination of Binary CSP and Boolean CSP. We attempt to assign values to variables, each of which has two possible values, to satisfy constraints on pairs of variables (Wikipedia definition). It can be solved in polynomial time.