Constraint Satisfaction

Definition: Constraint Satisfaction problems

Structured global path based search problems

- There are variables $X_1, X_2, ..., X_n$ - each variable X_i has a non-empty domain of values D_i
- There are constraints $C_1, C_2, ..., C_m$
  - Each constraint C_j involves some tuple of variables $(X_{i1}, X_{i2}, ..., X_{ik})$ called the scope
  - Specifies the allowable combinations of values for scope variables
  - C_j is usually given as a relation of k-tuples from $D_{i1} \times D_{i2} \times \ldots \times D_{ik}$ detailing these allowable simultaneous values
- An assignment associates a value from D_i to the variable X_i, for some or all of the variables
  - It is called **complete** if every variable is assigned a value
  - It is called **consistent** (or legal) if no constraint is violated
- A solution is a complete, consistent assignment
- Some CSPs require a solution to maximise/minimize some objective function defined on the set of arguments

1 Examples of CSPs

1.1 Graph 3-colourability problem (G3COL)

- Variable X_i for every vertex v_i in the given graph G
- Domain of values for each variable is {R,W,B} (Red, White, Blue)
- For every pair of variables (X_i, X_j) for which the corresponding vertices are joined by an edge $e = (v_i, v_j)$ in G
  - There is a constraint C_e with scope (X_i, X_j)
  - The tuples of simultaneous values in the constraint C_e consist of the set of all pairs of distinct colours

1.2 Satisfiability(SAT)

- Variable X_i foe every propositional variable X_i appearing in the given c.n.f formula $\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_m$
- Domain of values for each variable is {true,false}
- Constraint C_j for every clause ϕ_j
  - The scope of C_j is the set of propositional variables involved in ϕ_j
  - The tuples of simultaneous values in the constraint C_j are the truth assignments on the corresponding propositional variables making ϕ_j evaluate to true

2 CSPs as search problems

We can realise CSPs as "global path-based" search problems canonically

- A state in the state space is an assignment
- The initial state consists of the empty assignment
- The state x' is the successor (we assume only one action) of some state x if

  - x is consistent(we can move to inconsistent states, but once there, are stuck) and the assignment x' is the extension of x by the assignment of some value to some previously unassigned variable

- A goal state is a complete, consistent assignment (lies at depth n in the search tree where n is the number of variables)
- We have a constant step-cost of 1 per transition

We can also realise a CSP as a "local state-based" search problem, the same as above, but $f(x) = 1$ if x is a complete consistent assignment and $f(x) = 0$ otherwise

The CSP framework lends itself to the development of specific heuristic methods

- Not available in the general setting
- So that the inherent structure of a CSP leads to an "exponentially simplified" search algorithm (coming up)

3 Commutativity

The most crucial observation to make about CSPs - commutativity

- In order to come up with an assignment it does not matter in which order we choose to assign values to variables

Hence in every CSP search algorithm we expand a node of the search tree by considering possible assignments for only a single variable - it's up to us to choose which variable

Consequently we no longer speak of "the" search tree with CSPs but talk of "a" search tree, for we will build different search trees depending on which variable we choose to expand a node with respect to

We are essentially "pruning" the search tree of a vast number of branches but do not risk missing a goal-node if there is one

We also have leeway in the order in which we choose to assign variables - judicious orderings can yield improved performance

4 Real-world CSPs

In reality CSPs can be much more general e.g., domains of values need not be finite nor take discrete values - many real world problems have as their domains the natural numbers or real numbers

In the case of infinite domains of values, constraints might not be finitely describable, they might contain an infinite set of allowable value combinations

In such a circumstance we need to develop constraint languages

Definition: Constraint Languages

Languages that allow us to describe infinite sets of objects

Many real world constraint satisfaction problems involve preference constraints:

- Lectures A and B both prefer only teaching in the morning but are willing to teach in the afternoon

5 Back-tracing search for CSPs

Definition: Back-tracking search

A depth-first search w.r.t chosen orderings of both variables and values

Commutativity: we make assignments to one variable at a time so that

- Every search tree node that is constructed (apart from the root) is labelled with the assignment of some value to some variable
- The children of any node correspond to the assignment of a different value to the same variable

If we ever have some fringe node for which the assignment obtained by tracing the path from the fringe node violates a constraint (inconsistent) then this fringe node has no successors

So we obtain different search trees not only depending on the order in which we choose variables to assign values to but also depending upon the order in which we choose to assign values to a chosen variable

6 Example of a back-tracking search

Solve the following CSP

- Variables B,A,C with domain of values {1,2,3,4}
- Constraints:

$$(A, B) \in \{(1,2), (1,3), (1,4), (2,3), (2,4), (3,4)\}$$
$$(B, C) \in \{(1,2), (1,3), (1,4), (2,3), (2,4), (3,4)\}$$

Expand the nodes until there is an acceptable one

7 Back tracking algorithm

Listing 1 Backtrack-Search(CSP)

```
answer=Recursive-Backtrack(ass=∅)
```

Listing 2 Recursive-Backtrack(ass)

```
if ass is complete and consistent then return ass
if ass is consistent than
    var=Select-Variable(ass)
    for each value in Order-Domain(var,ass)
        ass = ass + {var=value}
        result = Recursive-Backtrack(ass)
        if result ≠ failure then
            return result
        else
            ass = ass - {var=value}
return failure
```

8 Minimum remaining values (MRV) heuristic

Can often do better than just leaving the functions Select-Variable and Order-Domain to be implementation dependent - we can employ heuristics to choose variables and values

Definition: Minimum remaining values heuristic

- Decides upon the next variable to assign a value to
- Intuition: find variable that are likely to ”cause problems” as soon as we can as otherwise we’ll have a lengthy backtrack to undertake
- Maintains, for every unassigned variable X, the number of legitimate values
- The next variable chosen has the smallest set of legitimate values with ties broken arbitrarily

Extreme scenario

- We are at point p in the execution and the set of legitimate values for X is empty
- Immediately attempt to assign a value to X, fail and recursively back-track
  - Otherwise, if we were to make alternative assignments from point p then at some future time we would still have to back-track all the way to point p

9 Least constraining value heuristic

Definition: Least constraining value heuristic

- Assists in the selection of a value to an already chosen variable X
- Any choice of value for X will, in general, rule out certain values for other unassigned variables (from their set of legitimate values)
- The total number of all such ruled-out values over all unassigned variables is maintained (in reality we just need to worry about those with unassigned variable sharing a constraint with X)
- Values for X are ordered in decreasing numbers of ruled-out values - ”illegal” values are not considered and ties are broken arbitrarily

Intuition - Try to leave the maximum flexibility for subsequent variable assignments

When we choose some value for our variable, we do not touch the sets of legitimate values for other unassigned variables even though we might know some of these values can be ruled out as this requires additional maintenance