Review: Constraint Satisfaction Problems

• How is a CSP defined?
• How do we solve CSPs?
Backtracking search algorithm

function Recursive-Backtracking (assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← Select-Unassigned-Variable(Variables[csp], assignment, csp)
  for each value in Order-Domain-Values(var, assignment, csp)
    if value is consistent with assignment given Constraints[csp]
      add \{var = value\} to assignment
      result ← Recursive-Backtracking(assignment, csp)
      if result ≠ failure then return result
      remove \{var = value\} from assignment
  return failure

• Improving backtracking efficiency:
  – Which variable should be assigned next?
  – In what order should its values be tried?
  – Can we detect inevitable failure early?
Which variable should be assigned next?

• **Most constrained variable:**
  – Choose the variable with the fewest legal values
  – A.k.a. **minimum remaining values (MRV)** heuristic
Which variable should be assigned next?

- **Most constrained variable:**
  - Choose the variable with the fewest legal values
  - A.k.a. *minimum remaining values* (MRV) heuristic
Which variable should be assigned next?

• **Most constraining variable:**
  – Choose the variable that imposes the most constraints on the remaining variables
  – Tie-breaker among most constrained variables
Which variable should be assigned next?

- **Most constraining variable:**
  - Choose the variable that imposes the most constraints on the remaining variables
  - Tie-breaker among most constrained variables
Given a variable, in which order should its values be tried?

• Choose the **least constraining value**:  
  – The value that rules out the fewest values in the remaining variables
Given a variable, in which order should its values be tried?

• Choose the **least constraining value**:  
  – The value that rules out the fewest values in the remaining variables

Which assignment for Q should we choose?

- Allows 1 value for SA
- Allows 0 values for SA
Early detection of failure: Forward checking

• Keep track of remaining legal values for unassigned variables
• Terminate search when any variable has no legal values
Early detection of failure: Forward checking

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values
Early detection of failure: Forward checking

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values
Early detection of failure: Forward checking

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values
Early detection of failure: Forward checking

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values
Constraint propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures.

- NT and SA cannot both be blue!

- **Constraint propagation** repeatedly enforces constraints locally.
Arc consistency

• Simplest form of propagation makes each pair of variables consistent:
  – $X \rightarrow Y$ is consistent iff for every value of $X$ there is some allowed value of $Y$
Arc consistency

• Simplest form of propagation makes each pair of variables consistent:
  – \( X \rightarrow Y \) is consistent iff for every value of \( X \) there is some allowed value of \( Y \)

• If \( X \) loses a value, all pairs \( Z \rightarrow X \) need to be rechecked
Arc consistency

- Simplest form of propagation makes each pair of variables consistent:
  - $X \rightarrow Y$ is consistent iff for every value of $X$ there is some allowed value of $Y$

- If $X$ loses a value, all pairs $Z \rightarrow X$ need to be rechecked
Arc consistency

• Simplest form of propagation makes each pair of variables consistent:
  – $X \rightarrow Y$ is consistent iff for every value of $X$ there is some allowed value of $Y$

• Arc consistency detects failure earlier than forward checking
• Can be run as a preprocessor or after each assignment
Arc consistency algorithm AC-3

function AC-3(csp) returns the CSP, possibly with reduced domains
inputs: csp, a binary CSP with variables \( \{X_1, X_2, \ldots, X_n\} \)
local variables: queue, a queue of arcs, initially all the arcs in csp

while queue is not empty
    \((X_i, X_j) \leftarrow \text{REMOVE-FIRST}(queue)\)
    if REMOVE-INCONSISTENT-VALUES\((X_i, X_j)\) then
        for each \(X_k\) in NEIGHBORS\([X_i]\) do
            add \((X_k, X_i)\) to queue

function REMOVE-INCONSISTENT-VALUES\((X_i, X_j)\) returns true iff succeeds
removed \(\leftarrow false\)
for each \(x\) in DOMAIN\([X_i]\)
    if no value \(y\) in DOMAIN\([X_j]\) allows \((x, y)\) to satisfy the constraint \(X_i \leftrightarrow X_j\)
        then delete \(x\) from DOMAIN\([X_i]\), removed \(\leftarrow true\)
return removed
Does arc consistency always detect the lack of a solution?

- There exist stronger notions of consistency (path consistency, k-consistency), but we won’t worry about them.
Backtracking search with inference

function Recursive-Backtracking\( (assignment, csp) \)
  if assignment is complete then return assignment
  \( var \leftarrow \text{Select-Unassigned-Variable}(\text{Variables}[csp], assignment, csp) \)
  for each value in Order-Domain-Values\( (var, assignment, csp) \)
    if value is consistent with assignment given Constraints\( [csp] \)
      add \( \{ var = value \} \) to assignment
      result ← Recursive-Backtracking\( (assignment, csp) \)
      if result ≠ failure then return result
      remove \( \{ var = value \} \) from assignment
  return failure

• Do inference (forward checking or constraint propagation) here
Local search for CSPs

• Hill-climbing, simulated annealing typically work with “complete” states, i.e., all variables assigned

• To apply to CSPs:
  – Allow states with unsatisfied constraints
  – Attempt to improve states by reassigning variable values

• Variable selection:
  – Randomly select any conflicted variable

• Value selection by min-conflicts heuristic:
  – Choose value that violates the fewest constraints
  – I.e., hill-climb with \( h(n) = \) total number of violated constraints

\[ \begin{align*}
&\begin{array}{|c|c|c|}
\hline
& \text{h = 5} & \text{h = 2} & \text{h = 0} \\
\hline
\end{array}
\end{align*} \]
Summary

• CSPs are a special kind of search problem:
  – States defined by values of a fixed set of variables
  – Goal test defined by constraints on variable values

• **Backtracking** = depth-first search where successor states are generated by considering assignments to a single variable
  – **Variable ordering** and **value selection** heuristics can help significantly
  – **Forward checking** prevents assignments that guarantee later failure
  – **Constraint propagation** (e.g., arc consistency) does additional work to constrain values and detect inconsistencies

• Local search can be done by iterative min-conflicts
CSP in computer vision: Line drawing interpretation

An example polyhedron:

Variables: edges

Domains: +, −, →, ←

Desired output:

D. Waltz, 1975
CSP in computer vision:
Line drawing interpretation

Four vertex types:

Constraints imposed by each vertex type:

D. Waltz, 1975
CSP in computer vision: 4D Cities

1. When was each photograph taken?
2. When did each building first appear?
3. When was each building removed?

Set of Photographs: 

Set of Objects: Buildings

CSP in computer vision: 4D Cities

- Goal: reorder images (columns) to have as few violations as possible
CSP in computer vision: 4D Cities

• **Goal:** reorder images (columns) to have as few violations as possible

• **Local search:** start with random ordering of columns, swap columns or groups of columns to reduce the number of conflicts

• Can also reorder the rows to group together points that appear and disappear at the same time – that gives you buildings
CSPs and NP-completeness

• The SAT problem
  – Given a formula in 3-CNF, find out whether there exists an assignment of the variables that makes it evaluate to true, e.g.:

\[
(X_1 \lor \overline{X}_7 \lor X_{13}) \land (\overline{X}_2 \lor X_{12} \lor X_{25}) \land \ldots
\]

• SAT is \textit{NP-complete} (Cook, 1971)
  – It’s in NP and every other problem in NP can be reduced to it
  – So are graph coloring, n-puzzle, and generalized sudoku
  – What are the implications of this for AI?