Constraint Satisfaction Problems

- Standard search problems:
  - State is a "black box": arbitrary data structure
  - Goal test can be any function over states
  - Successor function can also be anything

- Constraint satisfaction problems (CSPs):
  - A special subset of search problems
  - State is defined by variables \( x_i \) with values from a domain \( D_i \) sometimes \( D_i \) depends on \( i \)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables
  - Allows useful general-purpose algorithms with more power than standard search algorithms

Example: Map Coloring

- Variables:
- Domains: \( D \) = \{red, green, blue\}
- Constraints: adjacent regions must have different colors

Solution: WA, NT, Q, NSW, V, SA, T

Example: N-Queens

- Formulation 1:
  - Variables: \( X_{ij} \)
  - Domains: \{0, 1\}
  - Constraints:

  \[
  \forall i, j, k \quad (X_{ij}, X_{jk}, X_{ki}) \in \{(0,0),(0,1),(1,0)\}
  \sum_{i,j} X_{ij} = N
  \]

Example: N-Queens

- Formulation 2:
  - Variables: \( Q_k \)
  - Domains: \{1, 2, 3, \ldots, N\}
  - Constraints:
    - Non-threatening:
      \[
      \forall i, j \quad \text{non-threatening}(Q_i, Q_j)
      \]
    - Explicit:
      \[
      (Q_1, Q_2) \in \{(1,3),(1,4)\}
      \]

Constraint Graphs
Constraint Graphs

- Binary CSP: each constraint relates (at most) two variables
- Binary constraint graph: nodes are variables, arcs show constraints
- General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!

Example: Sudoku

- Variables: Each (open) square
- Domains: {1,2,...,9}
- Constraints:
  - 9-way alldiff for each column
  - 9-way alldiff for each row
  - 9-way alldiff for each region (or can have a bunch of pairwise inequality constraints)

Varieties of CSPs and Constraints

Varieties of CSPs

- Discrete Variables
  - Finite domains
    - Size $d$ means $O(d^n)$ complete assignments
    - E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)
  - Infinite domains (integers, strings, etc.)
    - E.g., job scheduling, variables are start/end times for each job
    - Linear constraints solvable, nonlinear undecidable
- Continuous variables
  - E.g., start/end times for Hubble Telescope observations
  - Linear constraints solvable in polynomial time by LP methods (see cs170 for a bit of this theory)

Varieties of Constraints

- Varieties of Constraints
  - Unary constraints involve a single variable (equivalent to reducing domains), e.g.:
    - SA vs. WA
  - Binary constraints involve pairs of variables, e.g.:
    - Higher-order constraints involve 3 or more variables:
      - e.g., cryptarithmetic column constraints
- Preferences (soft constraints):
  - E.g., red is better than green
  - Often representable by a cost for each variable assignment
  - Gives constrained optimization problems
  - We’ll ignore these until we get to Bayes’ nets

Real-World CSPs

- Assignment problems: e.g., who teaches what class
- Timetabling problems: e.g., which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Circuit layout
- Fault diagnosis
- … lots more!
- Many real-world problems involve real-valued variables…

Solving CSPs

- Standard search formulation of CSPs
  - States defined by the values assigned so far (partial assignments)
    - Initial state: the empty assignment, {} (or can have a bunch of pairwise inequality constraints)
  - Successor function: assign a value to an unassigned variable
  - Goal test: the current assignment is complete and satisfies all constraints
  - We’ll start with the straightforward, naïve approach, then improve it

Standard Search Formulation

Search Methods

- What would BFS do?
- What would DFS do?
- What problems does naïve search have?
Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs.
- Idea 1: One variable at a time
  - Variable assignments are commutative, so fix ordering
    - I.e., \([\text{WA} = \text{red} \text{ then } \text{NT} = \text{green}]\) same as \([\text{NT} = \text{green} \text{ then } \text{WA} = \text{red}]\)
  - Only need to consider assignments to a single variable at each step
- Idea 2: Check constraints as you go
  - I.e., consider only values which do not conflict previous assignments
  - Might have to do some computation to check the constraints
    - "Incremental goal test"
- Depth-first search with these two improvements is called backtracking search (not the best name)
- Can solve \(n\)-queens for \(n \approx 25\)

Backtracking Example

- Backtracking = DFS + variable-ordering + fail-on-violation

Backtracking Search

```python
function BacktrackingSearch(assignment) returns solution or failure
    return RecursiveBacktracking(assignment)

function RecursiveBacktracking(assignment) returns solution or failure
    if assignment is complete then return assignment
    for each variable \(v\) not yet assigned do
        if assignment consistent with \(\text{Constraints}(v, assignment)\) then
            success = RecursiveBacktracking(assignment + \(v\))
            if success then return success
        end if
    end for
    return failure
```

- Backtracking = DFS + variable-ordering + fail-on-violation