Constraint satisfaction problems (CSPs)

Standard search problem:
- state is a “black box” — any old data structure
  that supports goal test, eval, successor

CSP:
- state is defined by variables $V_i$ with values from domain $D_i$
- goal test is a set of constraints specifying
  allowable combinations of values for subsets of variables

Simple example of a formal representation language

Allows useful general-purpose algorithms with more power
than standard search algorithms
Example: 4-Queens as a CSP

Assume one queen in each column. Which row does each one go in?

**Variables** $Q_1, Q_2, Q_3, Q_4$

**Domains** $D_i = \{1, 2, 3, 4\}$

**Constraints**

$Q_i \neq Q_j$ (cannot be in same row)

$|Q_i - Q_j| \neq |i - j|$ (or same diagonal)

Translate each constraint into set of allowable values for its variables

E.g., values for $(Q_1, Q_2)$ are $(1, 3)$ $(1, 4)$ $(2, 4)$ $(3, 1)$ $(4, 1)$ $(4, 2)$

---

**Constraint graph**

*Binary CSP:* each constraint relates at most two variables

*Constraint graph:* nodes are variables, arcs show constraints

![Constraint graph diagram]
Example: Map coloring

Color a map so that no adjacent countries have the same color

Variables
- Countries $C_i$

Domains
- $\{\text{Red, Blue, Green}\}$

Constraints
- $C_1 \neq C_2$, $C_1 \neq C_5$, etc.

Constraint graph:

Real-world CSPs

Assignment problems
- e.g., who teaches what class

Timetabling problems
- e.g., which class is offered when and where?

Hardware configuration

Spreadsheets

Transportation scheduling

Factory scheduling

Floorplanning

Notice that many real-world problems involve real-valued variables
**Applying standard search**

Let’s start with the straightforward, dumb approach, then fix it.

States are defined by the values assigned so far.

*Initial state:* all variables unassigned

*Operators:* assign a value to an unassigned variable

*Goal test:* all variables assigned, no constraints violated

Notice that this is the same for all CSPs!

---

**Implementation**

CSP state keeps track of which variables have values so far.

Each variable has a domain and a current value.

```
datatype CSP-STATE
  components: UNASSIGNED, a list of variables not yet assigned
                ASSIGNED, a list of variables that have values

datatype CSP-VAR
  components: NAME, for i/o purposes
              DOMAIN, a list of possible values
              VALUE, current value (if any)
```

Constraints can be represented

- explicitly as sets of allowable values, or
- implicitly by a function that tests for satisfaction of the constraint.
Standard search applied to map-coloring

Complexity of the dumb approach

Max. depth of space $m = \text{??} \cdot n$ (number of variables)

Depth of solution state $d = \text{??} \cdot n$ (all vars assigned)

Search algorithm to use: depth-first

Branching factor $b = \text{??} \cdot \sum_{i} |D_i|$ (at top of tree)

This can be improved dramatically by noting the following:

1) Order of assignment is irrelevant so many paths are equivalent
2) Adding assignments cannot correct a violated constraint
Backtracking search

Use depth-first search, but
1) fix the order of assignment, \( b = |D_i| \) (can be done in the SUCCESSORS function)
2) check for constraint violations

The constraint violation check can be implemented in two ways:
1) modify SUCCESSORS to assign only values that are allowed, given the values already assigned
or 2) check constraints are satisfied before expanding a state

Backtracking search is the basic uninformed algorithm for CSPs
Can solve \( n \)-queens for \( n \approx 15 \)

Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

Simplified map-coloring example:

<table>
<thead>
<tr>
<th></th>
<th>RED</th>
<th>BLUE</th>
<th>GREEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Can solve \( n \)-queens up to \( n \approx 30 \)
Heuristics for CSPs

More intelligent decisions on
which value to choose for each variable
which variable to assign next

Given $C_1 = \text{Red}$, $C_2 = \text{Green}$, choose $C_3 = ??$

$C_3 = \text{Green}$: least-constraining-value

Given $C_1 = \text{Red}$, $C_2 = \text{Green}$, what next??

$C_5$: most-constrained-variable

Can solve $n$-queens for $n \approx 1000$
Iterative algorithms 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
- operators reassign variable values

Variable selection: randomly select any conflicted variable

*min-conflicts* heuristic:
- choose value that violates the fewest constraints
- i.e., hillclimb with $h(n) =$ total number of violated constraints

Summary

CSPs are a special kind of problem:
- states defined by values of a fixed set of variables
- goal test defined by constraints on variable values

Backtracking = depth-first search with
- 1) fixed variable order
- 2) only legal successors

Forward checking prevents assignments that guarantee later failure

Variable ordering and value selection heuristics help significantly