What is Search For?

- Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space
- Planning: sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics give problem-specific guidance
- Identification: assignments to variables
  - The goal itself is important, not the path
  - All paths at the same depth (for some formulations)
  - CSPs are a specialized class of identification problems

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 \( X \) with values from a domain \( D \) (sometimes \( D \) depends on \( i \))
  - 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

CSP Examples

Example: Map Coloring

- Variables: WA, NT, Q, NSW, V, SA, T
- Domains: \( D = \{ \text{red, green, blue} \} \)
- Constraints: adjacent regions must have different colors
  - Implicit: \( \text{WA} \neq \text{NT} \)
  - Explicit: \( (\text{WA}, \text{NT}) \in \{ (\text{red, green}), (\text{red, blue}), \ldots \} \)
- Solutions are assignments satisfying all constraints, e.g.:
  \[
  \{ \text{WA}=\text{red}, \text{NT}=\text{green}, \text{Q}=\text{red}, \text{NSW}=\text{green}, \text{V}=\text{red}, \text{SA}=\text{blue}, \text{T}=\text{green} \}
  \]
**Example: N-Queens**

- **Formulation 1:**
  - Variables: \(X_{ij}\)
  - Domains: \([0, 1]\)
  - Constraints:
    \[
    \forall i, j, k \ (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\} \\
    \forall i, j, k \ (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\} \\
    \forall i, j, k \ (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\} \\
    \forall i, j, k \ (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\} \\
    \sum_{i,j} X_{ij} = N
    \]

**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: Cryptarithmetic**

- Variables: \(F \ T \ U \ W \ R \ O \ X_1 \ X_2\)
- Domains: \([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]\)
- Constraints:
  \(\text{alldiff}(F, T, U, W, R, O)\)
  \(O + O = R + 10 \cdot X_1\)

**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)
Example: The Waltz Algorithm

- The Waltz algorithm is for interpreting line drawings of solid polyhedra as 3D objects.
- An early example of an AI computation posed as a CSP.

Approach:
- Each intersection is a variable.
- Adjacent intersections impose constraints on each other.
- Solutions are physically realizable 3D interpretations.

Varieties of CSPs and Constraints

Varieties of CSPs

- Discrete Variables
  - Finite domains
  - Size of means $O(n^k)$ 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.:
    
  - 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.

Real-World CSPs

- Scheduling problems: e.g., when can we all meet?
- Timetabling problems: e.g., which class is offered when and where?
- Assignment problems: e.g., who teaches what class?
- Hardware configuration.
- Transportation scheduling.
- Factory scheduling.
- Circuit layout.
- Fault diagnosis.
- ... lots more!

Solving CSPs

- Many real-world problems involve real-valued variables...
Standard Search Formulation

- Standard search formulation of CSPs
- States defined by the values assigned so far (partial assignments)
  - Initial state: the empty assignment, {}.
  - 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.

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.
  - Only need to consider assignments to a single variable at each step.
- Idea 2: Check constraints as you go.
  - 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.
- What are the choice points?

[Demo: coloring -- dfs]

[Demo: coloring -- backtracking]
Improving Backtracking

- General-purpose ideas give huge gains in speed
- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Filtering: Can we detect inevitable failure early?
- Structure: Can we exploit the problem structure?

Filtering

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment

Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment

Filtering: 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!
  - Why didn't we detect this yet?
  - Constraint propagation: reason from constraint to constraint

Consistency of A Single Arc

- An arc \( X \rightarrow Y \) is consistent iff for every \( x \) in the tail there is some \( y \) in the head which could be assigned without violating a constraint
- Forward checking: Enforcing consistency of arcs pointing to each new assignment

Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are consistent:
  - Important: If \( X \) loses a value, neighbors of \( X \) need to be rechecked!
  - Arc consistency detects failure earlier than forward checking
  - Can be run as a preprocessor or after each assignment
  - What's the downside of enforcing arc consistency?
Enforcing Arc Consistency in a CSP

Function `ARC_Consistency` returns the CSP possibly with reduced domain:

```plaintext
FRONTIER [X1, X2, ..., Xn]  
for each (X, X') in FRONTIER do
  if X is inconsistent with X' then
    remove X from X' domain
```

Runtime: $O(n^2 d^3)$, can be reduced to $O(n^2 d^2)$

... but detecting all possible future problems is NP-hard – why?

[Demo: CSP applet (made available by aispace.org) -- n-queens]

Limitations of Arc Consistency

After enforcing arc consistency:
- Can have one solution left
- Can have multiple solutions left
- Can have no solutions left (and not know it)

Arc consistency still runs inside a backtracking search!

Ordering

Variable Ordering: Minimum remaining values (MRV):
- Choose the variable with the fewest legal left values in its domain

Why min rather than max?
- Also called “most constrained variable”
- “Fail-fast” ordering

Ordering: Least Constraining Value

- Value Ordering: Least Constraining Value
  - Given a choice of variable, choose the least constraining value
  - i.e., the one that rules out the fewest values in the remaining variables
  - Note that it may take some computation to determine this! (E.g., rerunning filtering)

Why least rather than most?
- Combining these ordering ideas makes 1000 queens feasible

[Demo: coloring -- backtracking + AC + ordering]