Announcements

- Project 1 due Thursday
- Lecture videos reminder: don’t count on it
- Midterm
- Section: CSPs
  - Tue 3-4pm, 285 Cory
  - Tue 4-5pm, 285 Cory
  - Wed 11-noon, 285 Cory
  - Wed noon-1pm, 285 Cory
Today

- CSPs
- Efficient Solution of CSPs
  - Search
  - Constraint propagation
- Local Search

Example: Map-Coloring

- Variables: \( WA, NT, Q, NSW, V, SA, T \)
- Domain: \( D = \{ \text{red, green, blue} \} \)
- Constraints: adjacent regions must have different colors
  \( WA \neq NT \)
  \( (WA, NT) \in \{ (\text{red, green}), (\text{red, blue}), (\text{green, red}), \ldots \} \)
- Solutions are assignments satisfying all constraints, e.g.:
  \[ \{WA = \text{red}, NT = \text{green}, Q = \text{red}, NSW = \text{green}, V = \text{red}, SA = \text{blue}, T = \text{green} \} \]
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 (circles):
  \( F, T, U, W, R, O, X_1, X_2, X_3 \)
- Domains:
  \( \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \)
- Constraints (boxes):
  \( \text{alldiff}(F, T, U, W, R, O) \)
  \( O + O = R + 10 \cdot X_1 \)
  \( \ldots \)
Example: Sudoku

Variables:
- Each (open) square

Domains:
- \{1,2,\ldots,9\}

Constraints:
- 9-way alldiff for each column
- 9-way alldiff for each row
- 9-way alldiff for each region

Example: The Waltz Algorithm

- The Waltz algorithm is for interpreting line drawings of solid polyhedra
- An early example of a computation posed as a CSP

- Look at all intersections
- Adjacent intersections impose constraints on each other
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 state of a robot
- 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 (equiv. to shrinking domains):
    \[ SA \not= \text{green} \]
  - Binary constraints involve pairs of variables:
    \[ SA \not= WA \]
  - 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
- Floorplanning
- Fault diagnosis
- … lots more!

- Many real-world problems involve real-valued variables…

Standard Search Formulation

- Standard search formulation of CSPs (incremental)
- Let’s start with the straightforward, dumb approach, then fix it

- States are defined by the values assigned so far
  - 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

- Simplest CSP ever: two bits, constrained to be equal
Search Methods

- What does BFS do?
- What does DFS do?
  - [demo]
- What’s the obvious problem here?
- What’s the slightly-less-obvious problem?

Backtracking Search

- Idea 1: Only consider a single variable at each point
  - Variable assignments are commutative, so fix ordering
    - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
    - Only need to consider assignments to a single variable at each step
    - How many leaves are there?
- Idea 2: Only allow legal assignments at each point
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to figure out whether a value is ok
  - “Incremental goal test”
- Depth-first search for CSPs with these two improvements is called *backtracking search* (useless name, really)
  - [DEMO]
- Backtracking search is the basic uninformed algorithm for CSPs
- Can solve n-queens for \( n = 25 \)
Backtracking Search

A function `Backtracking-Search(csp)` returns a solution or failure:

```
function Backtracking-Search(csp) returns solution/failure
return Recursive-Backtracking({}, csp)
```

A function `Recursive-Backtracking(assignment, csp)` returns a solution or failure:

- If `assignment` is complete, return `assignment`.
- Select an unassigned variable `var`.
- For each `value` in the domain of `var`:
  - If `value` is consistent with `assignment` and given `csp`, add `{var = value}` to `assignment`.
  - Set `result` to `Recursive-Backtracking(assignment, csp)`.
  - If `result` is not failure, return `result`.
- Remove `{var = value}` from `assignment`.
- Return failure.

- What are the choice points?

Backtracking Example

1. Start with node with tightest constraints.
2. Most closed neighbors.
3. Domain has become small.
4. Min degree heuristic.
5. Remaining values heuristic.
### Improving Backtracking

- General-purpose ideas can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?
  - Can we take advantage of problem structure?

### Minimum Remaining Values

- Minimum remaining values (MRV):
  - Choose the variable with the fewest legal values

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

- Tie-breaker among MRV variables
- Degree heuristic:
  - Choose the variable participating in the most constraints on remaining variables
- Why most rather than fewest constraints?

Least Constraining Value

- Given a choice of variable:
  - Choose the least constraining value
  - The one that rules out the fewest values in the remaining variables
  - Note that it may take some computation to determine this!
- Why least rather than most?
- Combining these heuristics makes 1000 queens feasible
Forward Checking

- Idea: Keep track of remaining legal values for unassigned variables (using immediate constraints)
- Idea: Terminate when any variable has no legal values

Constraint Propagation

- Forward checking propagates information from assigned to adjacent unassigned variables, but doesn't detect more distant failures:
- NT and SA cannot both be blue!
- Why didn’t we detect this yet?
- *Constraint propagation* repeatedly enforces constraints (locally)
Arc Consistency

- Simplest form of propagation makes each arc consistent
  - $X \rightarrow Y$ is consistent iff for every value $x$ there is some allowed $y$.

If $X$ loses a value, neighbors of $X$ need to be rechecked!
- Arc consistency detects failure earlier than forward checking
- What’s the downside of arc consistency?
- Can be run as a preprocessor or after each assignment

Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$
- … but detecting all possible future problems is NP-hard – why?

\[\text{function AC-3}(csp) \text{ returns the CSP, possibly with reduced domains}\]
\[\text{inputs: csp, a binary CSP with variables } \{X_1, X_2, \ldots, X_n\}\]
\[\text{local variables: queue, a queue of arcs, initially all the arcs in csp}\]
\[\text{while queue is not empty do}\]
\[\quad (X_i, X_j) \leftarrow \text{REMOVE-FIRST}(queue)\]
\[\quad \text{if REMOVE-INCONSISTENT-VALUES}(X_i, X_j) \text{ then}\]
\[\quad \text{for each } X_k \text{ in Neighbors}[X_i] \text{ do}\]
\[\quad \text{add } (X_k, X_i) \text{ to queue}\]

\[\text{function REMOVE-INCONSISTENT-VALUES}(X_i, X_j) \text{ returns true iff succeeds}\]
\[\text{removed} \leftarrow \text{false}\]
\[\text{for each } x \text{ in Domain}[X_i] \text{ do}\]
\[\quad \text{if no value } y \text{ in Domain}[X_j] \text{ allows } (x, y) \text{ to satisfy the constraint } X_i \rightarrow X_j\]
\[\quad \text{then delete } x \text{ from Domain}[X_i]; \text{ removed} \leftarrow \text{true}\]

\[\text{return removed}\]
Limitations of Arc Consistency

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

K-Consistency

- Increasing degrees of consistency
  - 1-Consistency (Node Consistency): Each single node’s domain has a value which meets that node’s unary constraints
  - 2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other
  - K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the kth node.

- Higher k more expensive to compute
Strong K-Consistency

- Strong k-consistency: also k-1, k-2, ..., 1 consistent
- Claim: strong n-consistency means we can solve without backtracking!
- Why?
  - Choose any assignment to any variable
  - Choose a new variable
  - By 2-consistency, there is a choice consistent with the first
  - Choose a new variable
  - By 3-consistency, there is a choice consistent with the first 2
  - ...
- Lots of middle ground between arc consistency and n-consistency! (e.g. path consistency)