Announcements

- Assignment due on Monday 11.59pm
- Email search.py and searchAgent.py to me
- Next week’s classes taught by Prof. Geiger

Today

- Efficient Solution of CSPs

Reminder: CSPs

- CSPs:
  - Variables
  - Domains
  - Constraints
    - Implicit (provide code to compute)
    - Explicit (provide a subset of the possible tuples)
  - Unary Constraints
  - Binary Constraints
  - N-ary Constraints

Example: N-Queens

- Formulation 2:
  - Variables: \( Q_k \)
  - Domains: \{1, 2, 3, \ldots N\}
  - Constraints:
    \[(Q_2, Q_3) \in \{(1, 3), (1, 4), \ldots \}\]
    \[
    \ldots
    \]

Example: Map-Coloring

- Variables: \( WA, NT, Q, NSW, V, SA, T \)
- Domain: \( D = \{\text{red, green, blue}\} \)
- Constraints: adjacent regions must have different colors
  - \( WA \neq NT \)
  - \((WA, NT) \in \{(\text{red, green}), (\text{red, blue}), (\text{green, red})\} \)
- Solutions are assignments satisfying all constraints, e.g.:
  - \( WA = \text{red}, NT = \text{green}, Q = \text{red}, NSW = \text{green}, V = \text{red}, SA = \text{blue}, T = \text{green} \)
Search Overview

- Basic solution: DFS / backtracking
  - Add a new assignment
  - Filter by checking for immediate violations
- Ordering:
  - Heuristics to choose variable order (MRV)
  - Heuristics to choose value order (LCV)
- Filtering:
  - Pre-filter unassigned domains after every assignment
  - Forward checking: remove values which immediately conflict with current assignments (makes MRV easy!)
  - Arc consistency - propagate indirect consequences of assignments

Backtracking Search

- Backtracking = DFS + var-ordering + fail-on-violation
- What are the choice points?

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?

Ordering: 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

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?
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?

Ordering: 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

Filtering: Forward Checking

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

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 propagates from constraint to constraint

Consistency of An Arc

- An arc \( X \rightarrow Y \) is consistent if 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 each arc pointing to the new assignment
Arc Consistency of a CSP

- A simple form of propagation makes sure all arcs are consistent:
  - WA
  - NT
  - Q
  - NSW
  - V
  - GA
  - T

- 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 enforcing arc consistency?
- Can be run as a preprocessor or after each assignment

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)

What went wrong here?

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
- (You need to know the k=2 algorithm)

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)

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?
Problem Structure

- Tasmania and mainland are independent subproblems
- Identifiable as connected components of constraint graph
- Suppose each subproblem has $c$ variables out of $n$ total
  - Worst-case solution cost is $O(n/c)(d^c)$, linear in $n$
  - $2^{20} = 4$ billion years at 10 million nodes/sec
  - $(4)(2^{20}) = 0.4$ seconds at 10 million nodes/sec

Tree-Structured CSPs

- Choose a variable as root, order variables from root to leaves such that every node’s parent precedes it in the ordering
  - For $i = n/2$, apply RemoveInconsistent( Parent($X_i$), $X_i$ )
  - For $i = 1 : n$, assign $X_i$ consistently with Parent($X_i$)
- Runtime: $O(n d^2)$ (why?)

Why does this work?
- Claim: After each node is processed leftward, all nodes to the right can be assigned in any way consistent with their parent.
- Proof: Induction on position

Why doesn’t this algorithm work with loops?
- Note: we’ll see this basic idea again with Bayes’ nets

Nearly Tree-Structured CSPs

- Conditioning: instantiate a variable, prune its neighbors’ domains
- Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree
- Cutset size $c$ gives runtime $O((d^c)(n-c)d^2)$, very fast for small $c$

Tree Decompositions

- Create a tree-structured graph of overlapping subproblems, each is a mega-variable
- Solve each subproblem to enforce local constraints
- Solve the CSP over subproblem mega-variables using our efficient tree-structured CSP algorithm

Tree-Structured CSPs

- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n d^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$
  - This property also applies to probabilistic reasoning (later): an important example of the relation between syntactic restrictions and the complexity of reasoning.
Iterative Algorithms for CSPs

- Local search methods typically work with “complete” states, i.e., all variables assigned

- To apply to CSPs:
  - Start with some assignment with unsatisfied constraints
  - Operators reassign variable values
  - No fringe! Live on the edge.

- 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 \( h(n) = \text{total number of violated constraints} \)

Example: 4-Queens

- States: 4 queens in 4 columns (\( 4^4 = 256 \) states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: \( c(n) = \text{number of attacks} \)

Performance of Min-Conflicts

- Given random initial state, can solve \( n \)-queens in almost constant time for arbitrary \( n \) with high probability (e.g., \( n = 10,000,000 \))

- The same appears to be true for any randomly-generated CSP except in a narrow range of the ratio

\[ R = \frac{\text{number of constraints}}{\text{number of variables}} \]

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 with one legal variable assigned per node
  - Variable ordering and value selection heuristics 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
  - Constraint graphs allow for analysis of problem structure
  - Tree-structured CSPs can be solved in linear time
  - Iterative min-conflicts is usually effective in practice