Lecture 17

CS 6380 Artificial Intelligence

Search and CSPs

Department of Computer Science and Engineering

Indian Institute of Technology Madras (IITM), India

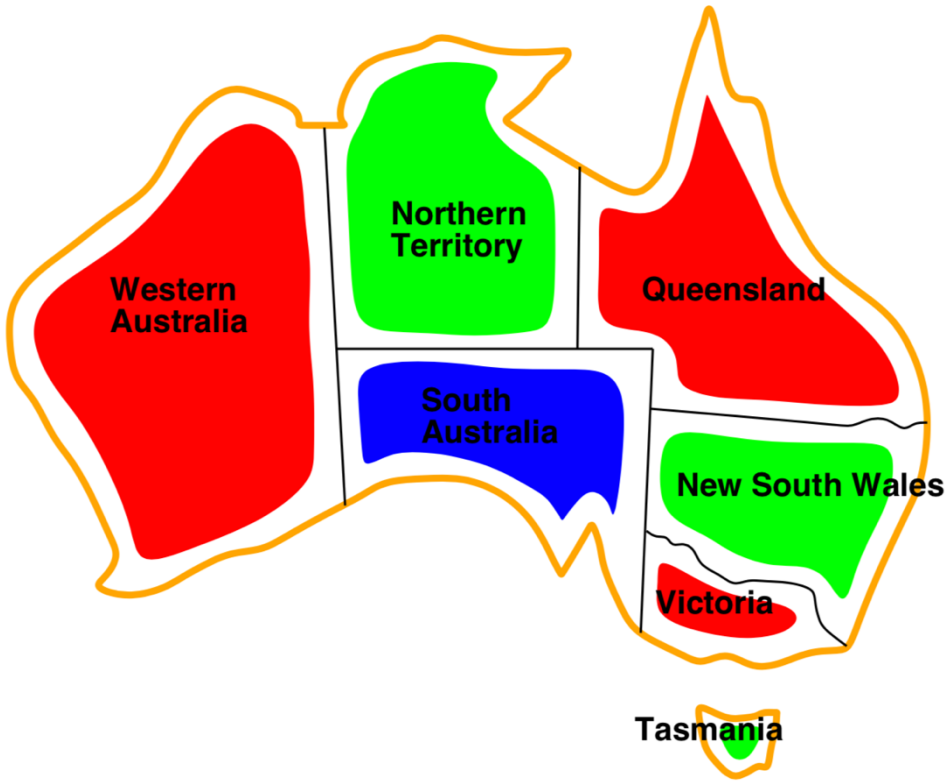


Material based on lecture slides for CSE 571: Artificial Intelligence by Siddharth Srivastava, ASU.

Recap

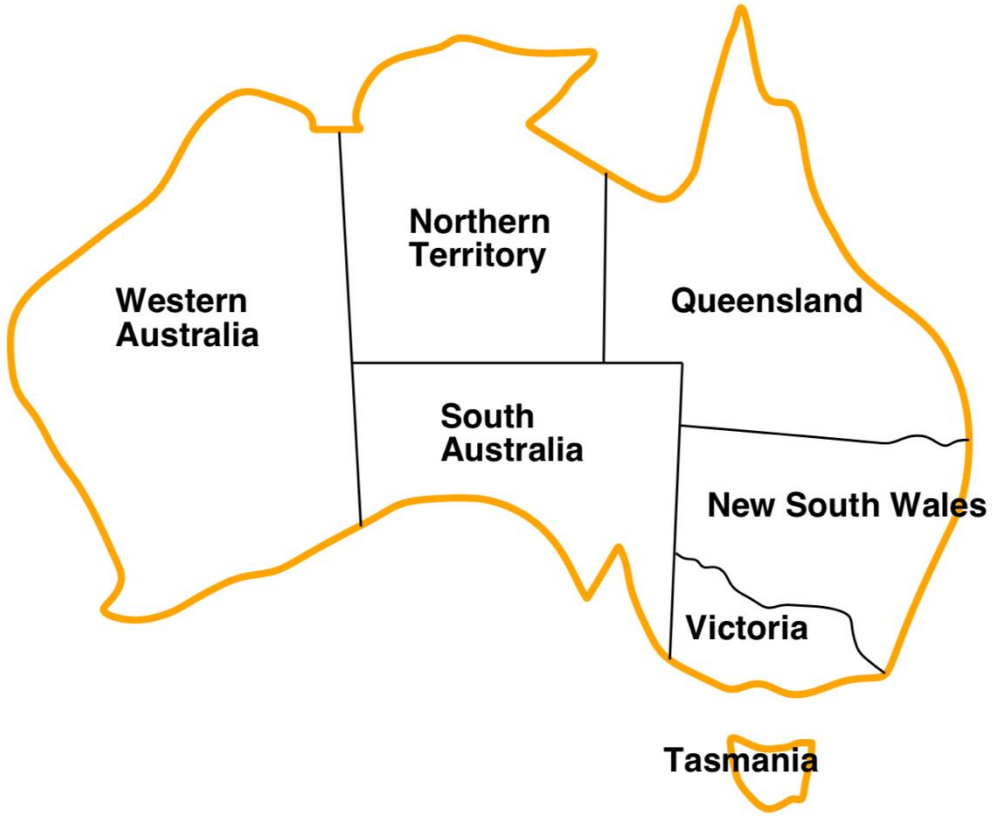
Map Coloring

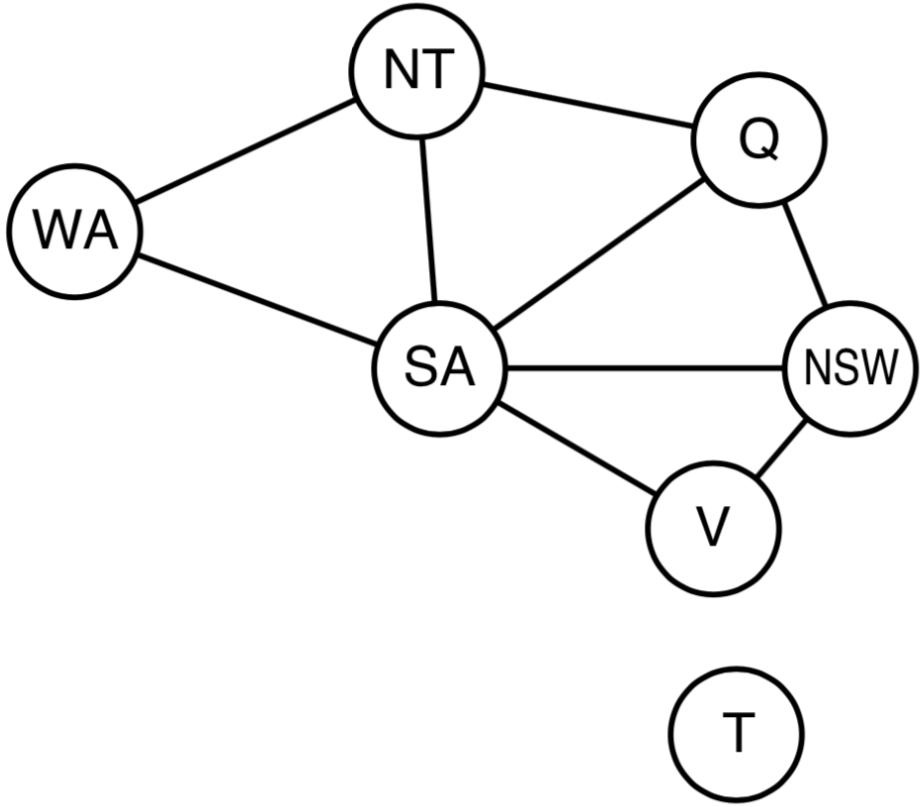
- Problem: color this map such that
  - adjacent regions have different colors
- Colors: red, green, blue
- Can formulate as:
  - Regions WA, NT, Q, NSW, SA, T
  - Possible colors: $red, green, blue$
- Compute:
  - A color mapping $\sigma: Region \rightarrow Color$ such that
  - For all adjacent r_1, r_2, $\sigma(r_1) \neq \sigma(r_2)$



Constraint Graph

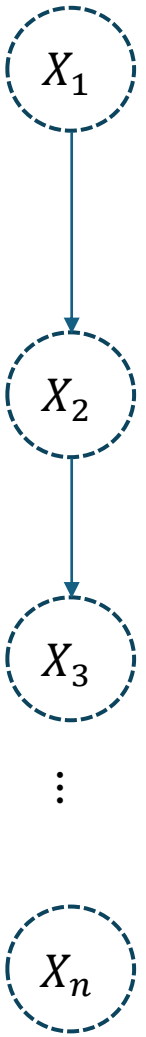
- Vertices: variables
- Edges between any pair of variables that are involved in a constraint





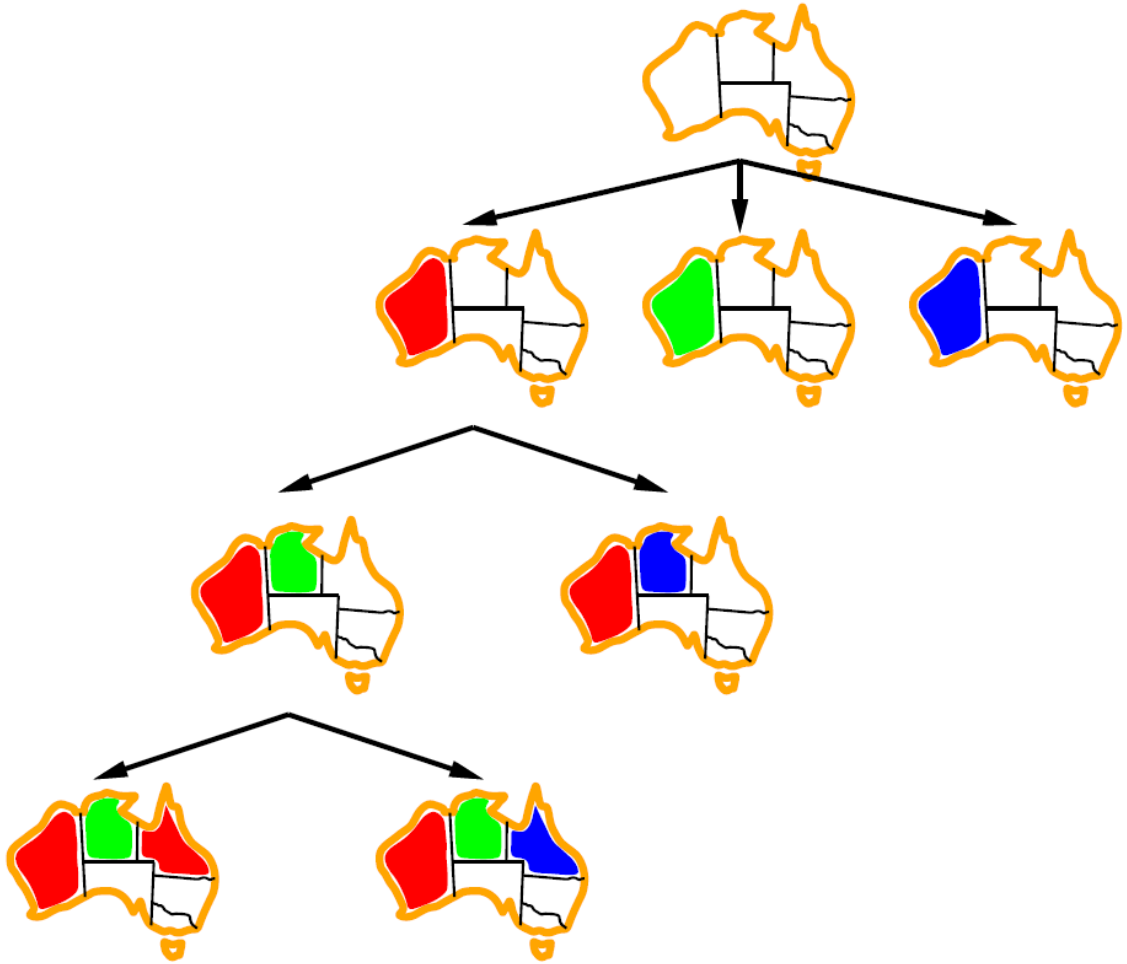
Backtracking Search

- Observation: variable assignments are commutative
  - Order of assignment doesn't matter
  - Can force each "level" to consider assignments to a single variable
  - Makes $b = D$; D^n leaves
- Backtracking search: DFS for CSPs, making a single variable assignment at each level
  - Baseline uninformed search algorithm for CSPs



Backtracking Search

- Observation: variable assignments are commutative
  - Order of assignment doesn't matter
  - Can force each "level" to consider assignments to a single variable
  - Makes $b = D$; D^n leaves
- Backtracking search: DFS for CSPs, making a single variable assignment at each level
  - Baseline uninformed search algorithm for CSPs



Optimizing Backtracking Search

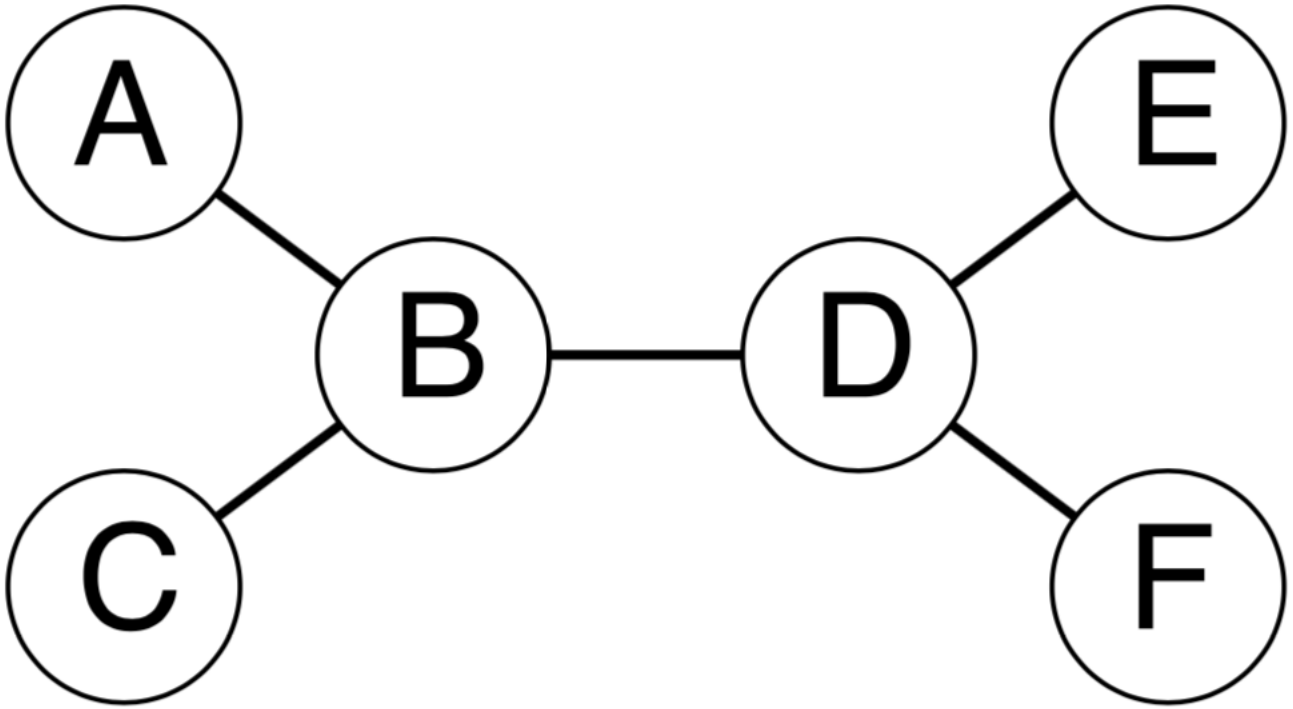
- As we noted, there's a lot more "structure" here. Can we exploit it?

- Ideas:
  - Better selection of the next variable to consider
  - Better ordering of its values for assignment
  - It would be great if we could…
    - Detect failure early (before a complete assignment)
    - Exploit the structure of the constraint graph

Optimizing Backtracking Search

- Some considerations to improve speed:
  - Which variable should be assigned next? (MRV)
  - In what order should its values be tried? (LCV)
  - Can failure be detected early (before a complete assignment)? (FC, Arc Consistency)
  - Can we exploit the structure of the constraint graph?

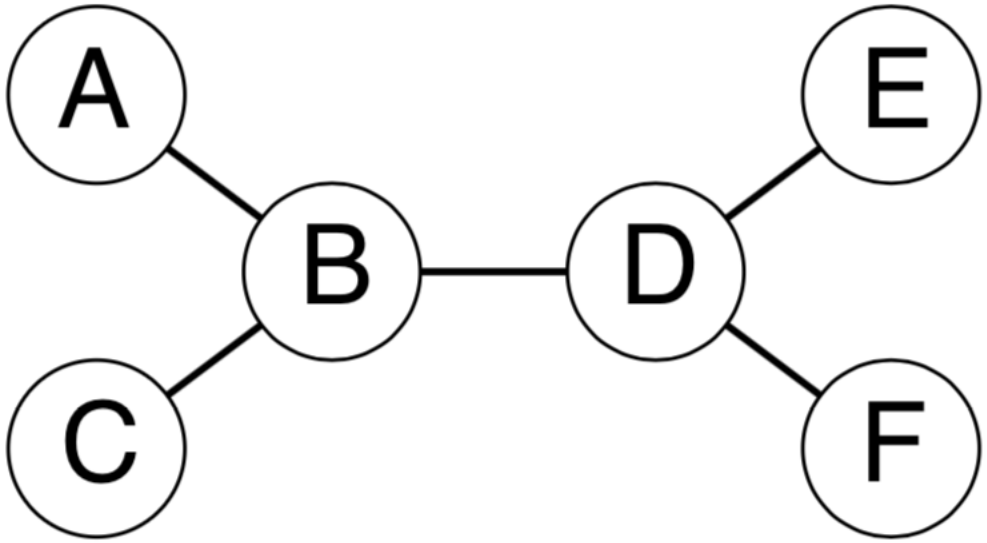
Tree-Structured CSPs

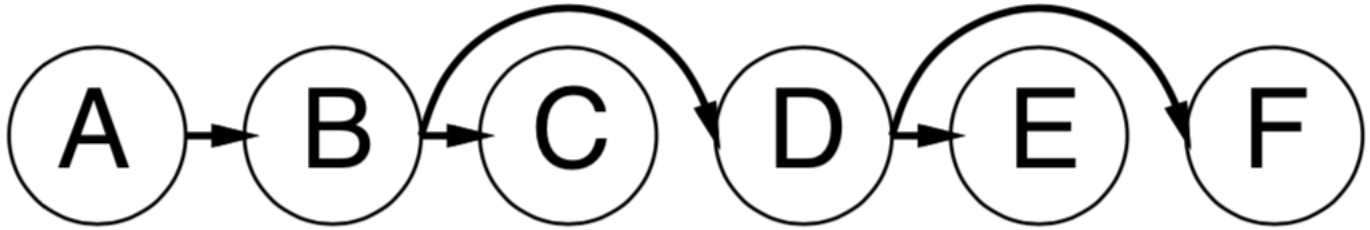


- If the constraint graph has no loops, the CSP can be solved in $O(nd^2)$ time
  - General CSPs have worst case time $O(d^n)$

Solving Tree-Structured CSPs

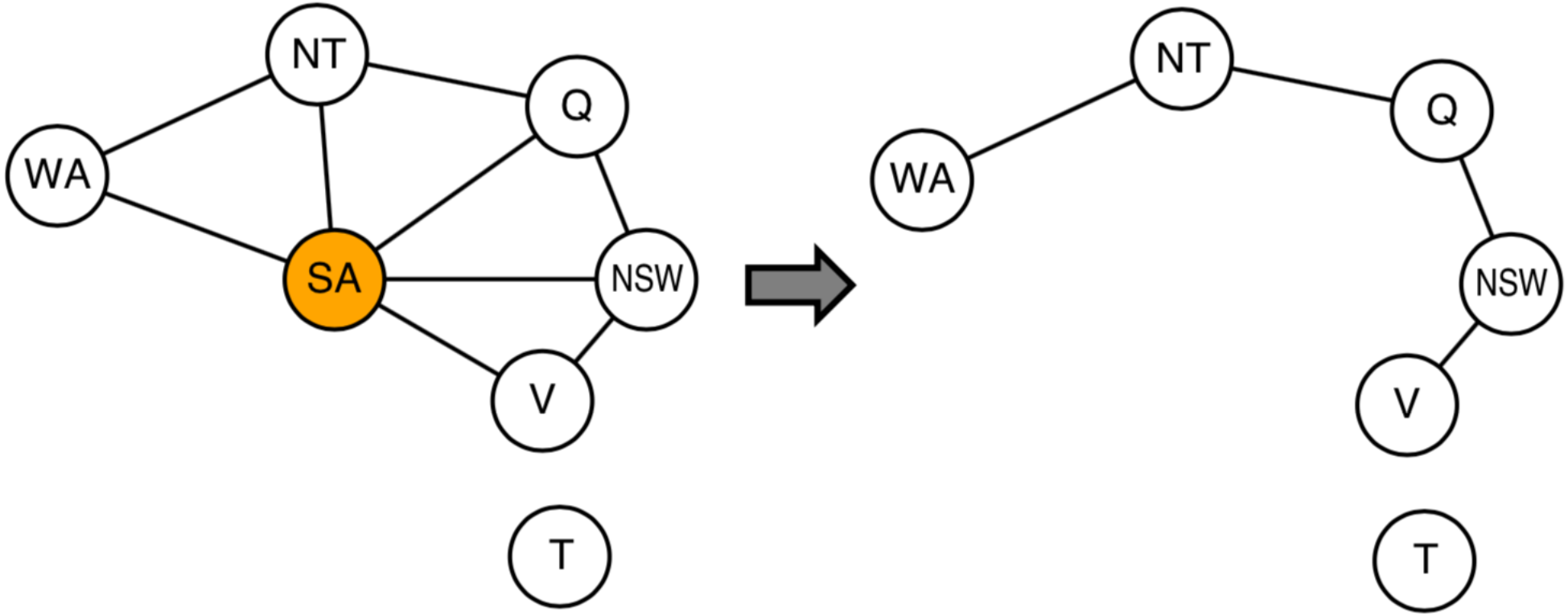
- Choose a root
- Order variables such that parent nodes precede children
- For j from n down to 2, apply RemoveInconsistent$(Parent(X_j), X_j)$
- For j from 1 to n assign X_j to make it consistent with $Parent(X_j)$





Almost Tree-Structured CSPs

- Conditioning: assign a variable, prune its neighbors' domains to generate a tree



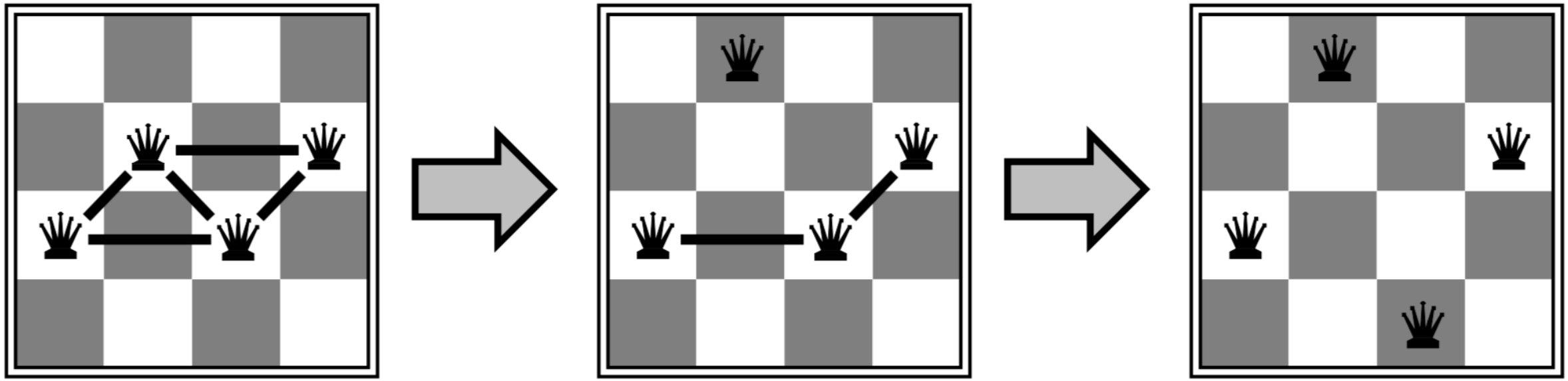
- Cutset conditioning: select a set to remove. Condition on all possible assignments
  - Cutset size c $\rightarrow$ time complexity $O(d^c(n-c)d^2)$

Local Search and CSPs

Local Search Driven by Iterative Improvement

- Recall, CSPs are a special case of search problems
  - Path to the solution is irrelevant
  - We care only about creating a goal state
- Iterative improvement idea:
  - Start with any non-goal state (full assignment that violates constraints)
  - Make small perturbations, see if you can find a goal state

N-queens



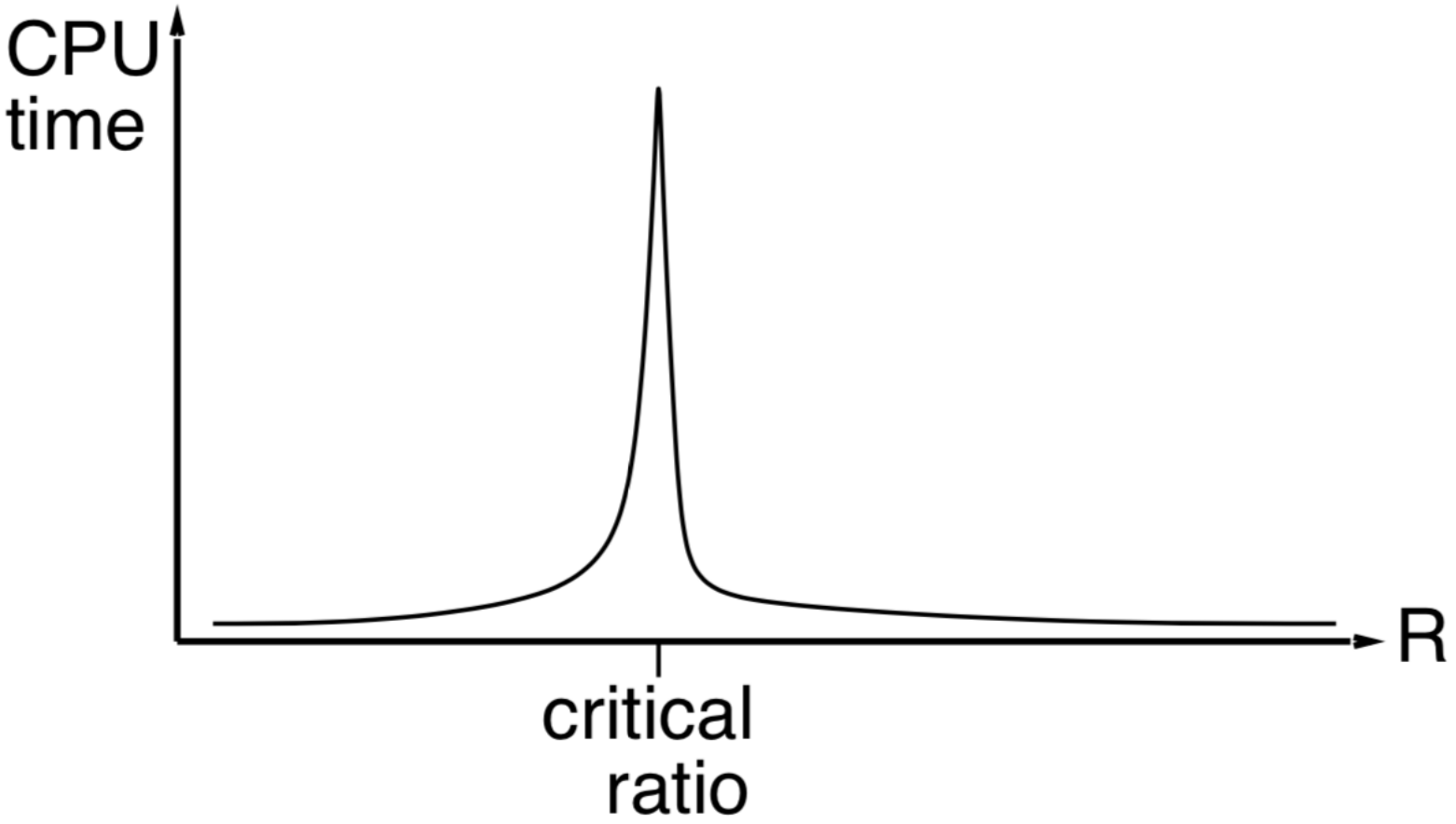
- Move a queen to reduce the number of violated constraints

Local Search for CSPs: Min-Conflicts

- Use hill-climbing or simulated annealing
  - Use complete (constraint violating) assignments as states
- Optimize on variable selection, value selection
- E.g., use min-conflicts heuristic for value selection
  - Choose value that violates fewest constraints for that variable

(as noted earlier)
Can solve n-queens almost instantaneously for large n (1 million)!

Performance of Min Conflicts for Randomly Generated CSPs



$$R = \frac{\#constraints}{\#variables}$$

Monte Carlo Tree Search

Beyond Minimax

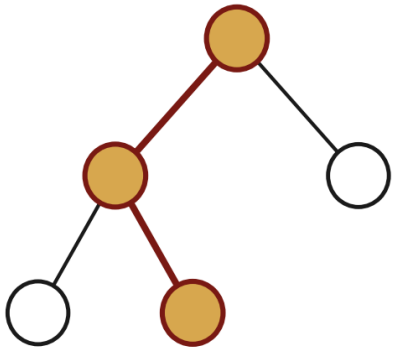
- Minimax with alpha beta pruning leans on two crutches
  - A branching factor small enough to search several plies deep
  - An evaluation function that scores cutoff states sensibly
- Some games deny us both
  - Go has roughly 250 legal moves per position
  - No simple board features reliably predict the winner in Go
- A different idea
  - Estimate the value of a position by playing many random games from it
  - The fraction of wins tells us how promising the position is

The Core Idea of MCTS

- Evaluate positions by simulation instead of a handcrafted function
  - Play out a complete game with a rollout policy and record the result
- Average the outcomes of many playouts
  - The average win rate estimates the value of a node
- Grow the search tree gradually and asymmetrically
  - Promising moves receive more simulations and deeper subtrees
  - Weak moves are still visited occasionally so nothing is ruled out too early
- Needs only the rules of the game and a source of randomness

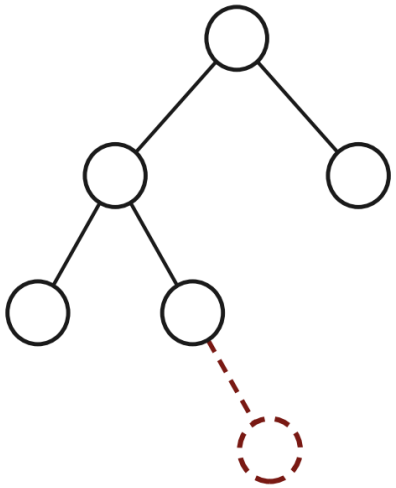
The Four Phases of MCTS

1 Selection



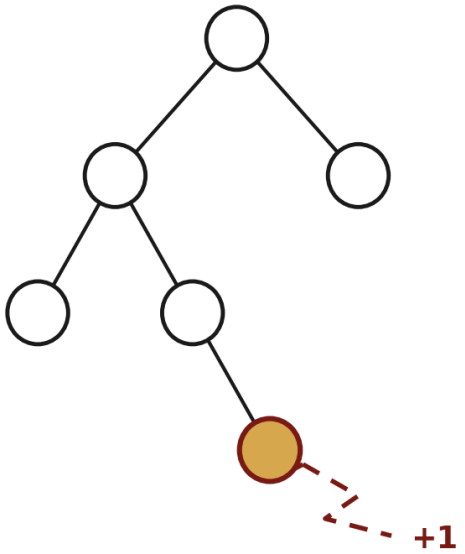
follow the tree policy
to a promising leaf

2 Expansion



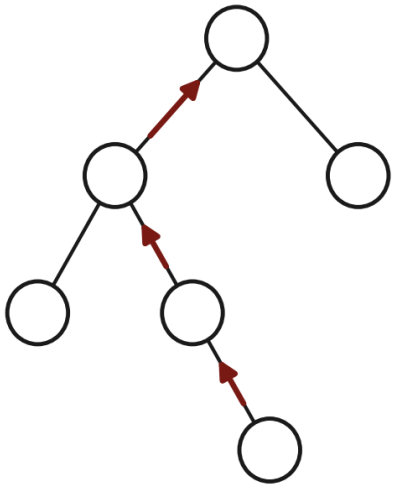
add one new child
to the tree

3 Simulation



play a random game
to the end

4 Backpropagation



send the result back
up the path

- Repeat all four phases until the time budget runs out, then play the most visited move at the root