

CS 188: Artificial Intelligence

Constraint Satisfaction Problems II

University of California, Berkeley



Today

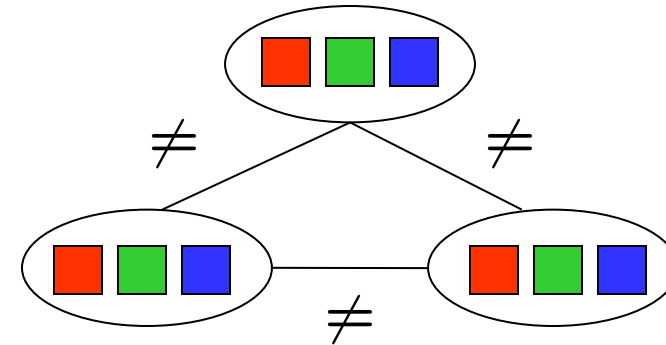
- Efficient Solution of CSPs
 - Ordering
 - Filtering
 - Structure



Reminder: CSPs

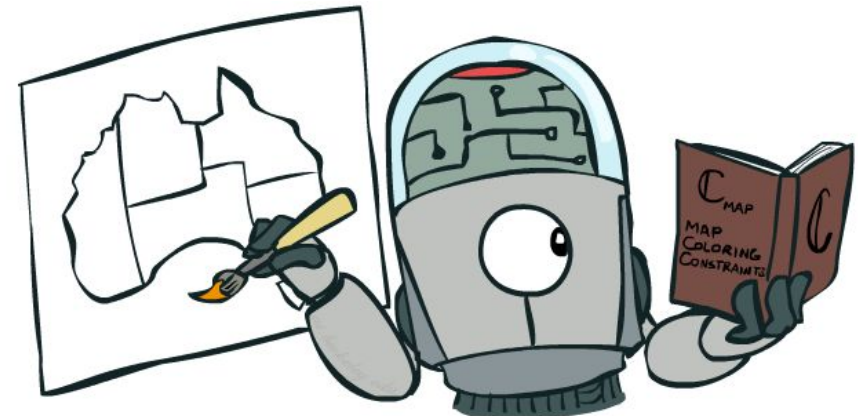
- CSPs:

- Variables
- Domains
- Constraints
 - Implicit (provide code to compute)
 - Explicit (provide a list of the legal tuples)
 - Unary / Binary / N-ary



- Goals:

- Here: find any solution
- Also: find all, find best, etc.



Backtracking Search

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment given CONSTRAINTS[csp] then
      add {var = value} to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove {var = value} from assignment
  return failure
```

Improving Backtracking

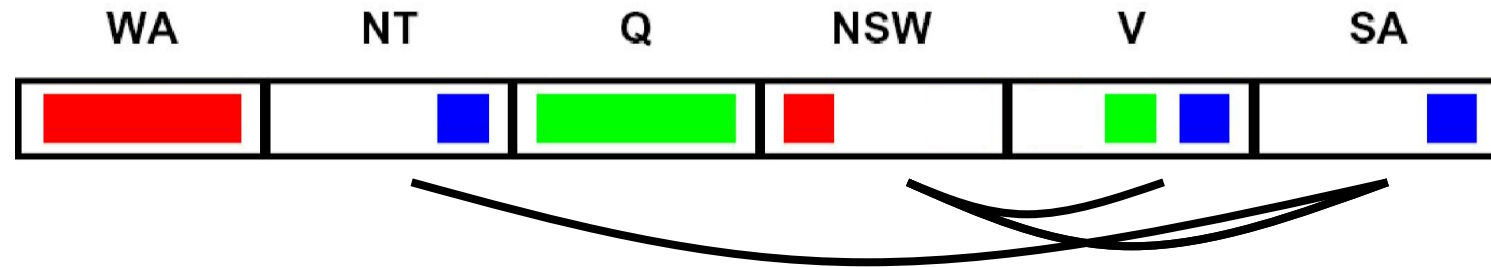
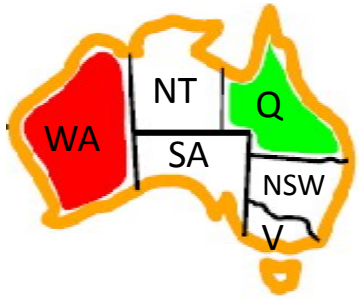
- General-purpose ideas give huge gains in speed
 - ... but it's all still NP-hard
- Filtering: Can we detect inevitable failure early?
- Ordering:
 - Which variable should be assigned next? (MRV)
 - In what order should its values be tried? (LCV)
- Structure: Can we exploit the problem structure?



Arc Consistency and Beyond

Arc Consistency of an Entire CSP

- 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 (head = the arrow!)
- A simple form of propagation makes sure **all** arcs are consistent:

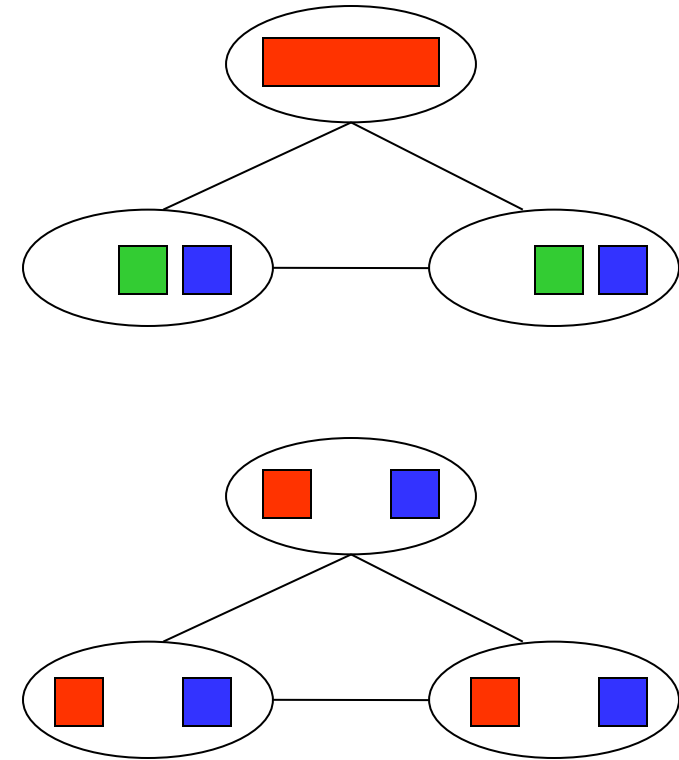


- Arc consistency detects failure earlier than forward checking
- Important: If X loses a value, neighbors of X need to be rechecked!
- Must rerun after each assignment!

*Remember:
Delete from
the tail!*

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!



What went wrong here?

Enforcing Arc Consistency in a CSP

```
function AC-3(csp) returns the CSP, possibly with reduced domains
inputs: csp, a binary CSP with variables  $\{X_1, X_2, \dots, X_n\}$ 
local variables: queue, a queue of arcs, initially all the arcs in csp

while queue is not empty do
     $(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\textit{queue})$ 
    if REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) then
        for each  $X_k$  in NEIGHBORS[ $X_i$ ] do
            add  $(X_k, X_i)$  to queue

```

```
function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) returns true iff succeeds
    removed  $\leftarrow$  false
    for each  $x$  in DOMAIN[ $X_i$ ] do
        if no value  $y$  in DOMAIN[ $X_j$ ] allows  $(x, y)$  to satisfy the constraint  $X_i \leftrightarrow X_j$ 
            then delete  $x$  from DOMAIN[ $X_i$ ]; removed  $\leftarrow$  true
    return removed

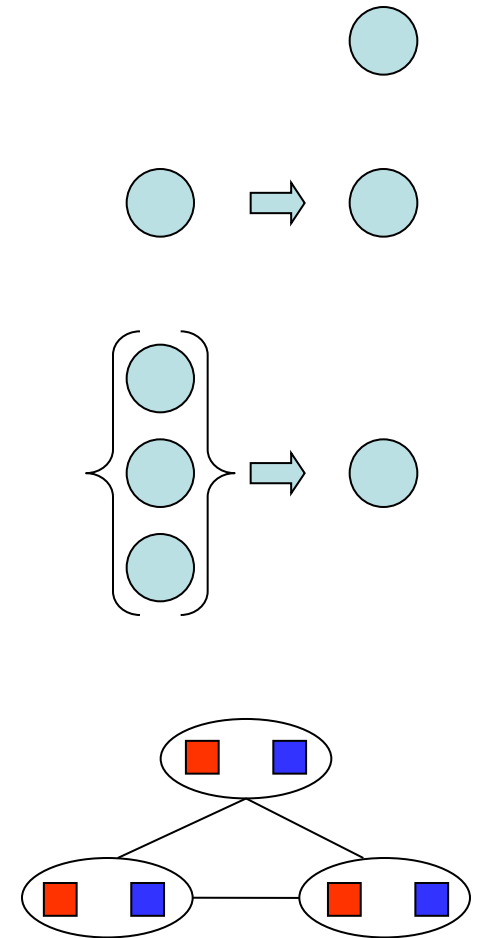
```

- Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$
- ... but detecting all possible future problems is NP-hard – why? Because it would enable us to solve NP-hard problems (like satisfiability).

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

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
 - The way we said this before: for any assignment to the tail, there is an assignment to the head.
 - 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 (but makes filtering better: tradeoff!)
- (You need to know the k=2 case: arc consistency)



Strong K-Consistency

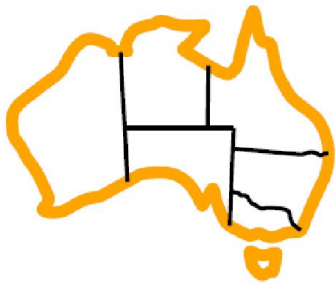
- Strong k-consistency: also k-1, k-2, ... 1 consistent
- Claim: strong n-consistency means we can solve without backtracking!
- Why?
 - Choose any assignment to any variable
 - Choose a new variable
 - By 2-consistency, there is a choice consistent with the first
 - Choose a new variable
 - By 3-consistency,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

Ordering

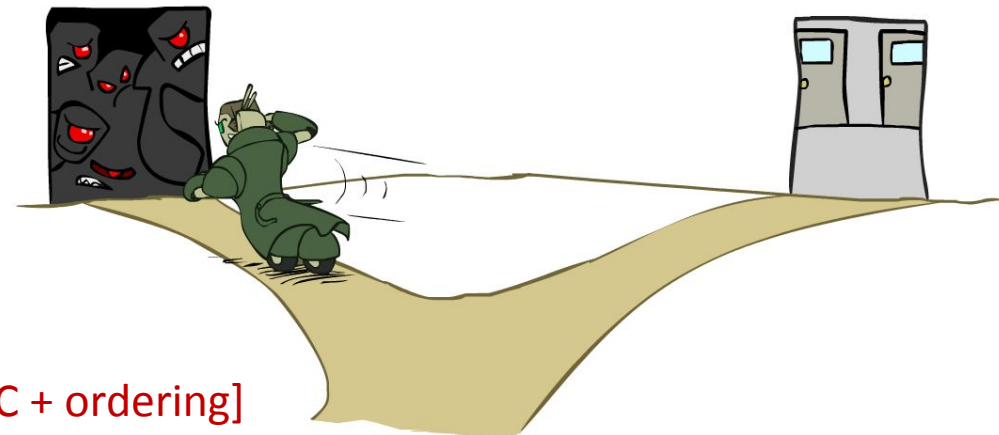


Ordering: Minimum Remaining Values

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



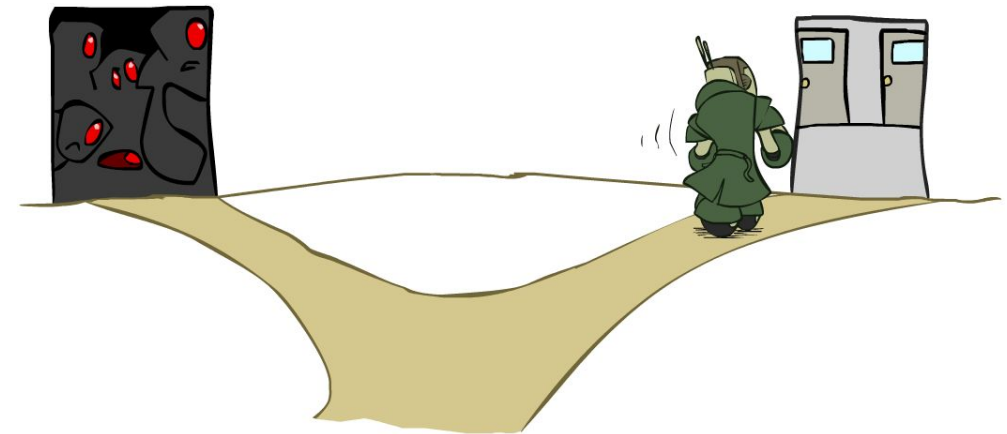
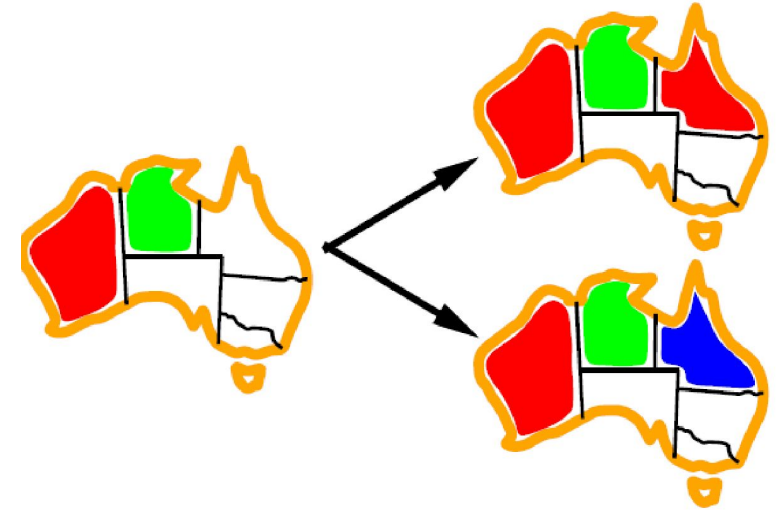
- Why min rather than max?
- Also called “most constrained variable”
- “Fail-fast” ordering



[Demo: coloring – backtracking + AC + 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

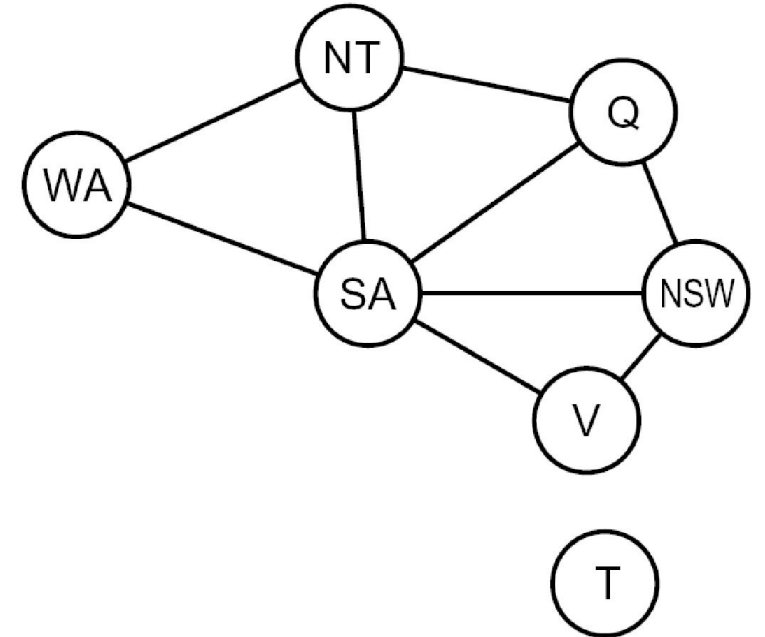


Structure

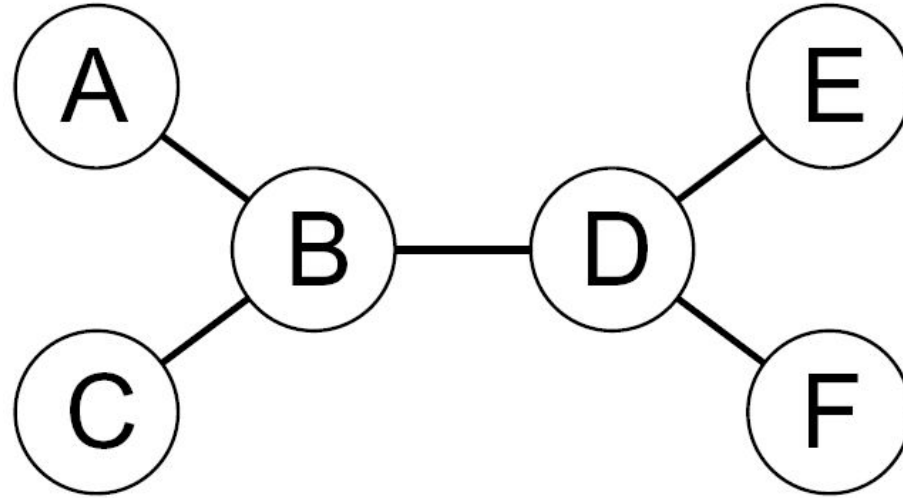


Problem Structure

- Extreme case: independent subproblems
 - Example: Tasmania and mainland do not interact
- Independent subproblems are identifiable as connected components of constraint graph
- Suppose a graph of n variables can be broken into subproblems of only c variables:
 - Worst-case solution cost is $O((n/c)(d^c))$, linear in n
 - E.g., $n = 80$, $d = 2$, $c = 20$
 - $2^{80} = 4$ billion years at 10 million nodes/sec
 - $(4)(2^{20}) = 0.4$ seconds at 10 million nodes/sec
- Rarely however do graphs decompose into perfectly disconnected pieces



Tree-Structured CSPs

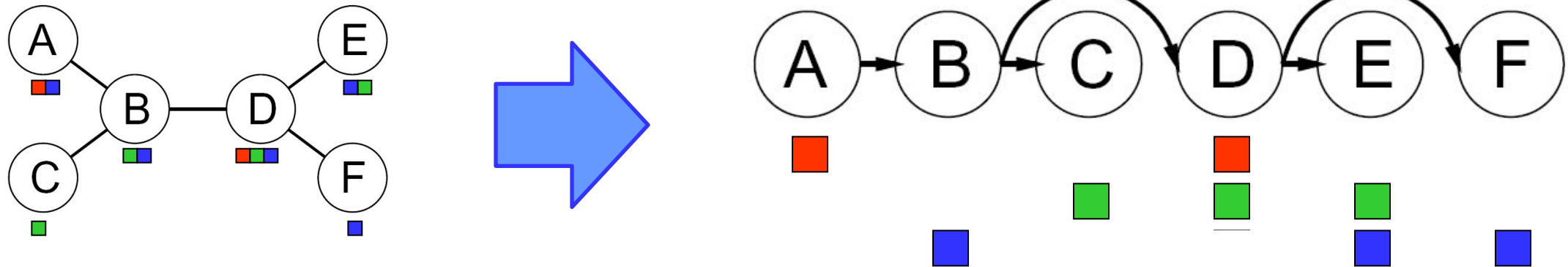


- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n d^2)$ time
 - Compare to general CSPs, where worst-case time is $O(d^n)$
- This property also applies to probabilistic reasoning (later): an example of the relation between syntactic restrictions and the complexity of reasoning

Tree-Structured CSPs

- Algorithm for tree-structured CSPs:

- Order: Choose a root variable, order variables so that parents precede children

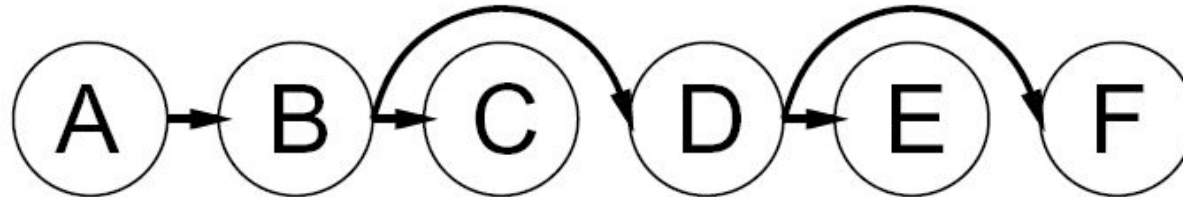


- Remove backward: For $i = n : 2$, apply $\text{RemoveInconsistent}(\text{Parent}(X_i), X_i)$
- Assign forward: For $i = 1 : n$, assign X_i consistently with $\text{Parent}(X_i)$

- Runtime: $O(n d^2)$ (why?)

Tree-Structured CSPs

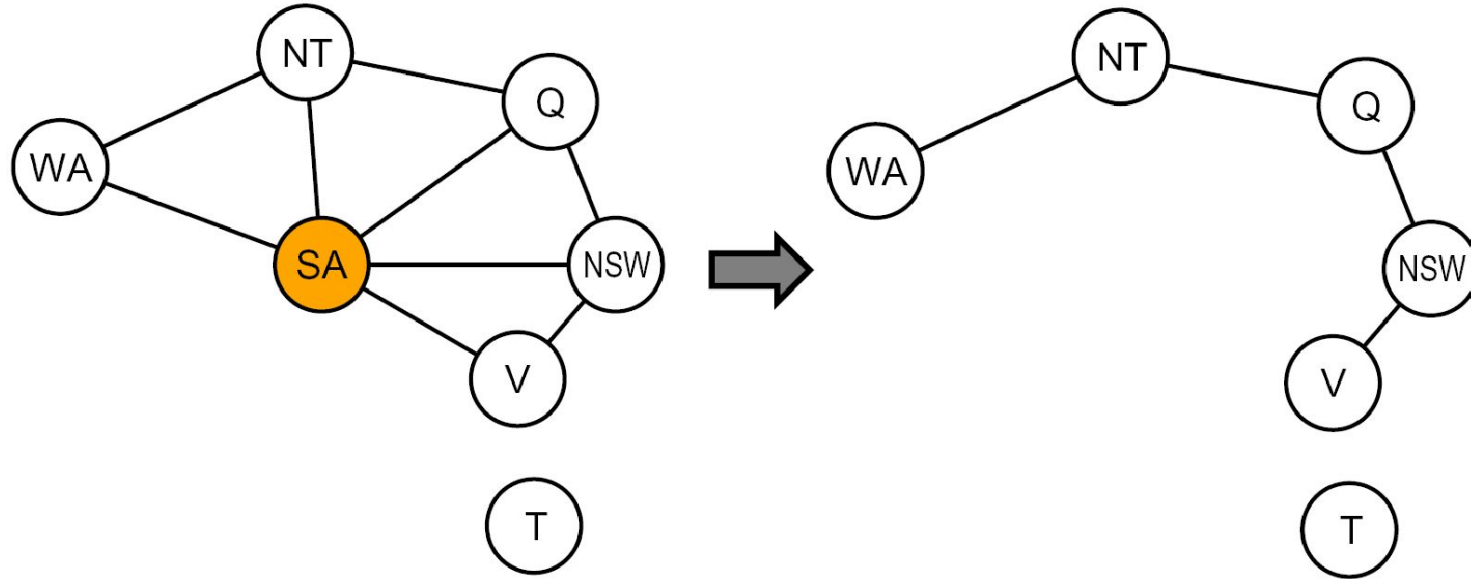
- Claim 1: After backward pass, all root-to-leaf arcs are consistent
- Proof: Each $X \rightarrow Y$ was made consistent at one point and Y 's domain could not have been reduced thereafter (because Y 's children were processed before Y)
 - Eg: once $B \rightarrow D$ is made consistent, we won't delete anything further from D 's domain (which could cause problems) because we're processing in the other direction



- Claim 2: If root-to-leaf arcs are consistent, forward assignment will not backtrack
- Proof: Induction on position
 - Whatever we assign to A , we know there's an assignment to B ; whatever we assign to $A+B$, we know there's an assignment to C
- Why doesn't this algorithm work with cycles in the constraint graph?
 - If two arcs point to the same node, they are guaranteed to both have individually consistent assignments – but those assignments might conflict. (Two parents might argue!)
- Note: we'll see this basic idea again with Bayes' nets

Improving Structure

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

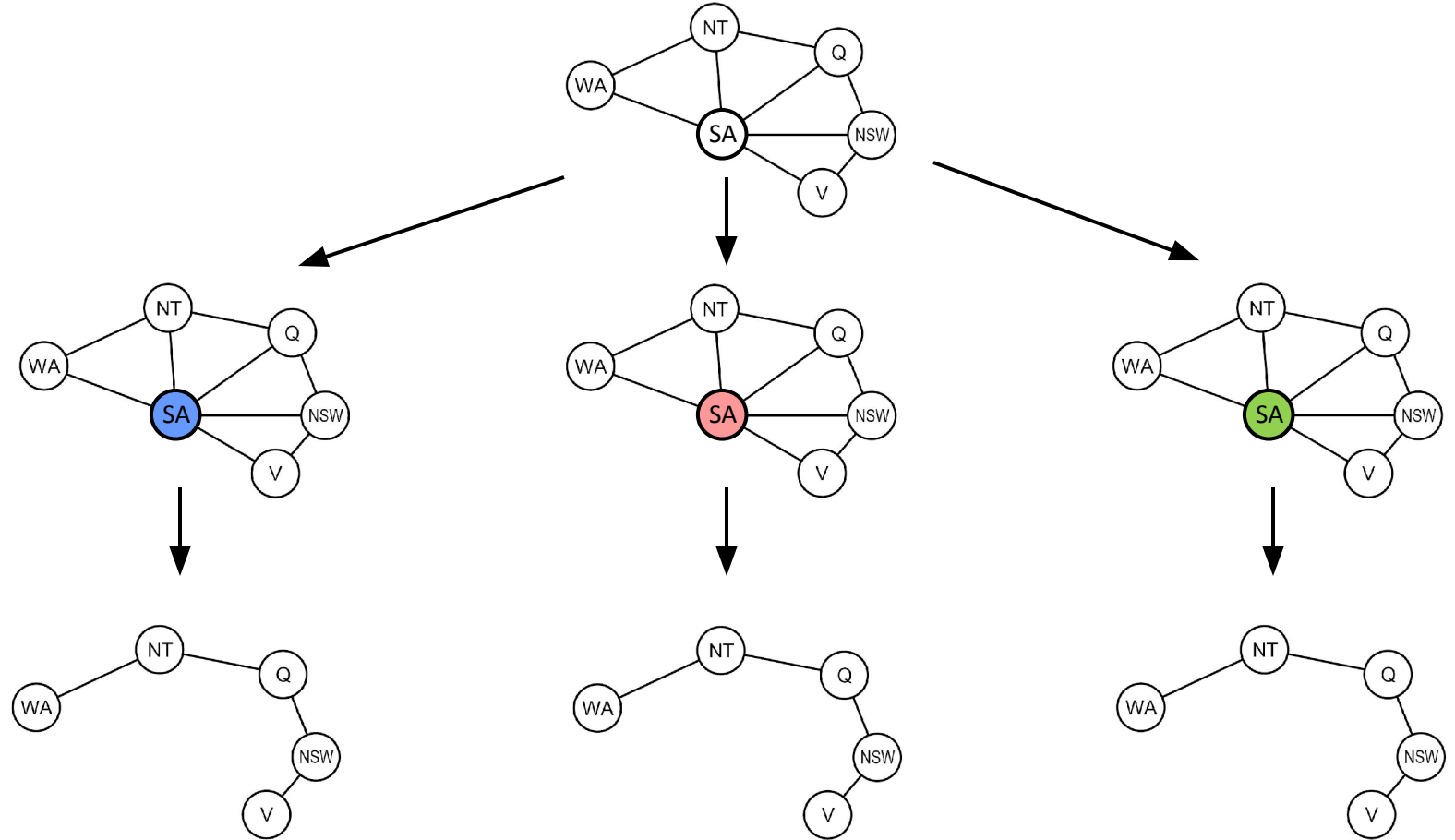
Cutset Conditioning

Choose a cutset

Instantiate the cutset
(all possible ways)

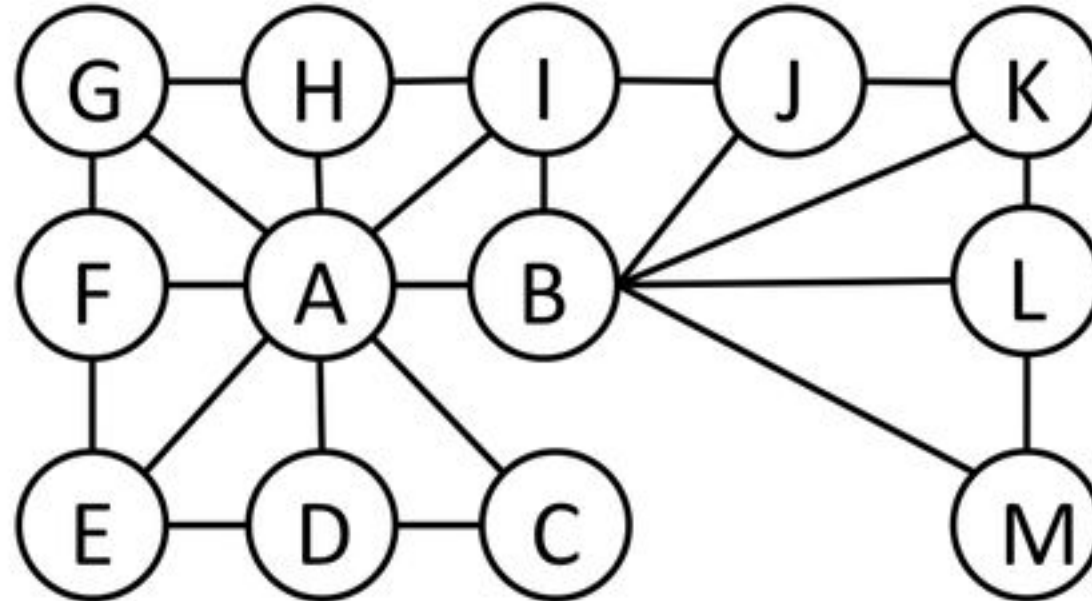
Compute residual CSP
for each assignment

Solve the residual CSPs
(tree structured)



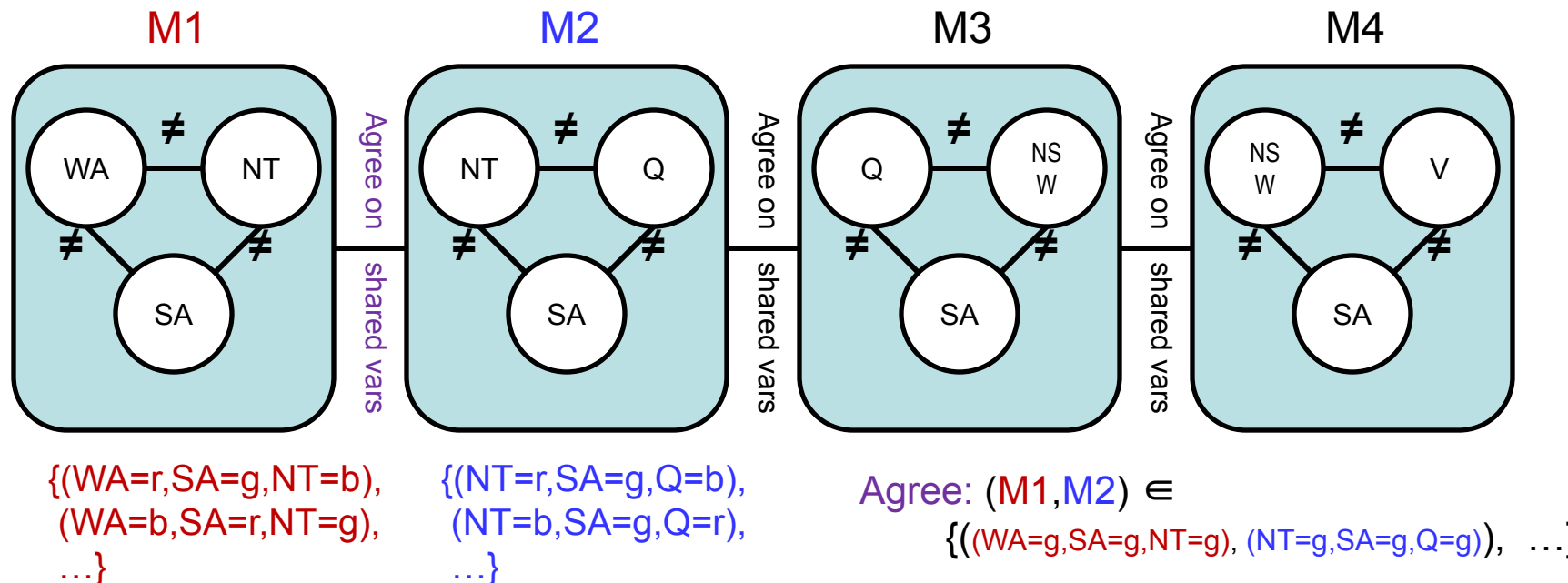
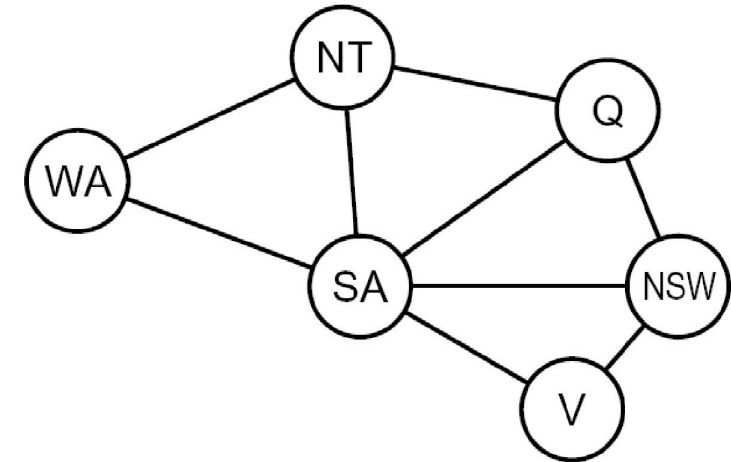
Cutset Quiz

- Find the smallest cutset that gives you a tree for the graph below.

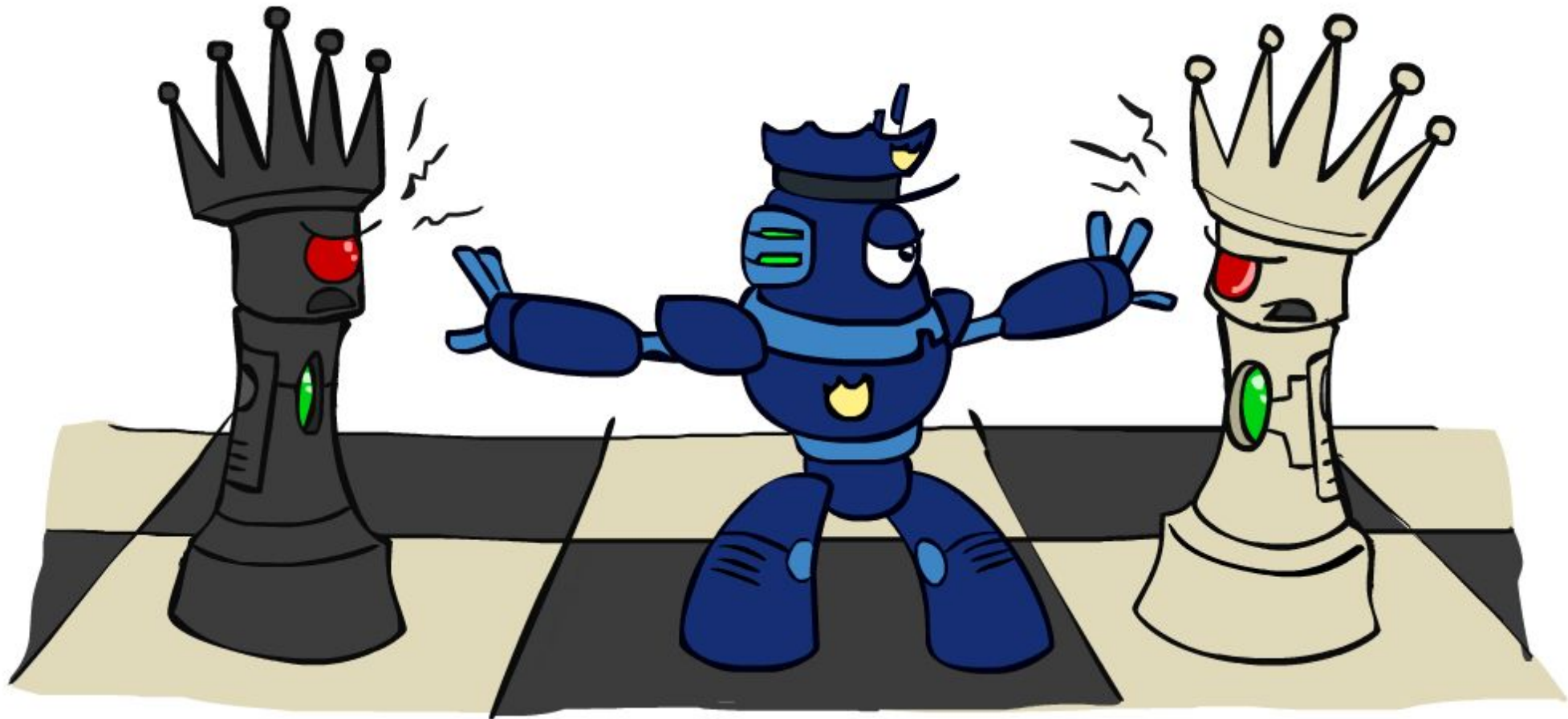


Tree Decomposition*

- Idea: create a tree-structured graph of mega-variables
- Each mega-variable encodes part of the original CSP
- Subproblems overlap to ensure consistent solutions
 - (can't just solve them separately)
- Have to set up subproblems correctly to ensure they are a tree (will not discuss how to do this)

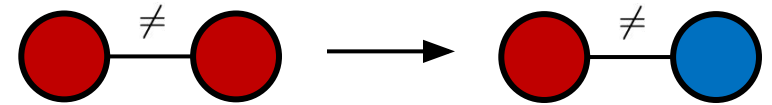


Iterative Improvement

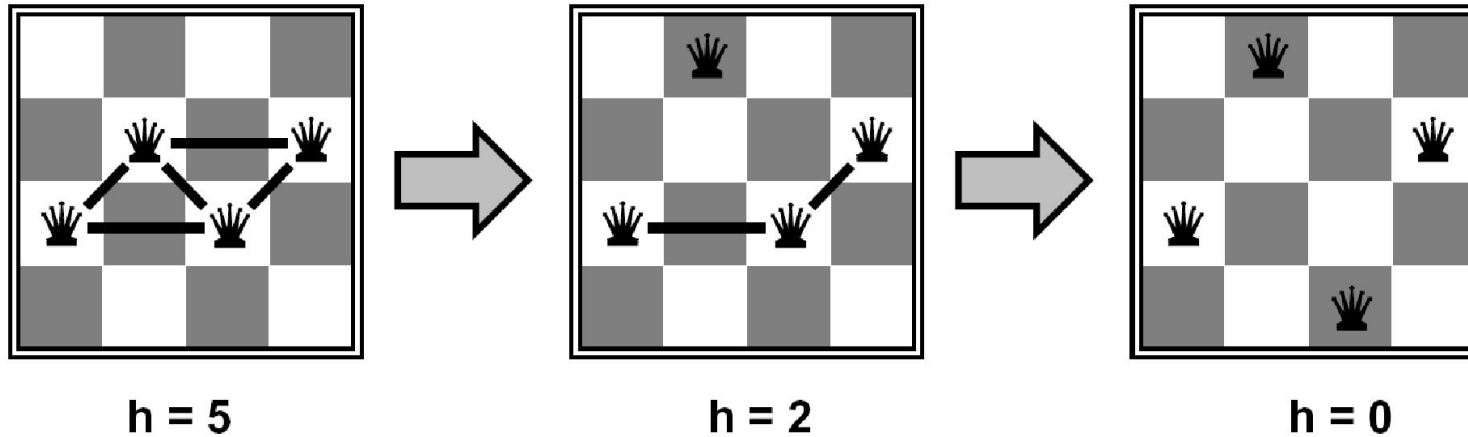


Iterative Algorithms for CSPs

- Local search methods typically work with “complete” states, i.e., all variables assigned
- To apply to CSPs:
 - Take an assignment with unsatisfied constraints
 - Operators *reassign* variable values
 - No fringe! Live on the edge.
- Algorithm: While not solved,
 - Variable selection: randomly select any conflicted variable
 - Value selection: min-conflicts:
 - Choose a value that violates the fewest constraints
 - I.e., hill climb with $h(n)$ = 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: $c(n) =$ number of attacks

Video of Demo Iterative Improvement – n Queens



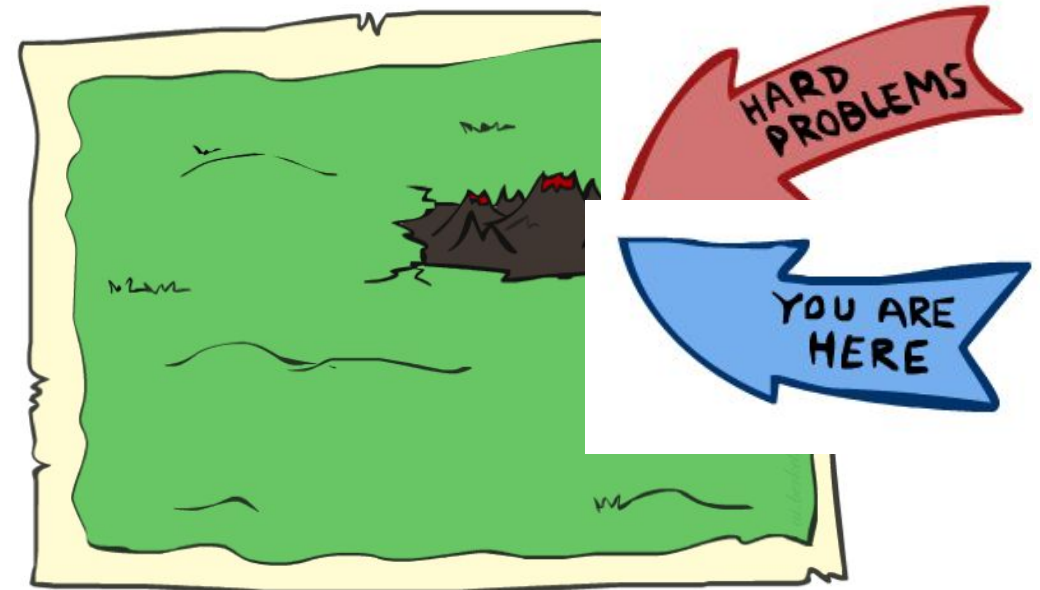
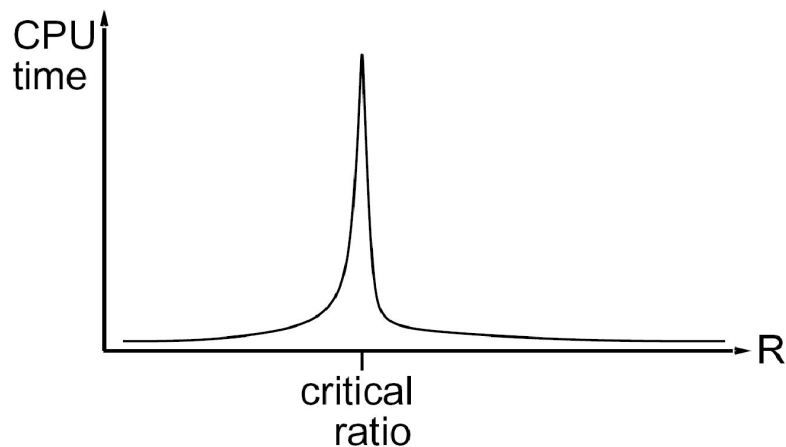
Video of Demo Iterative Improvement – Coloring



Performance of Min-Conflicts

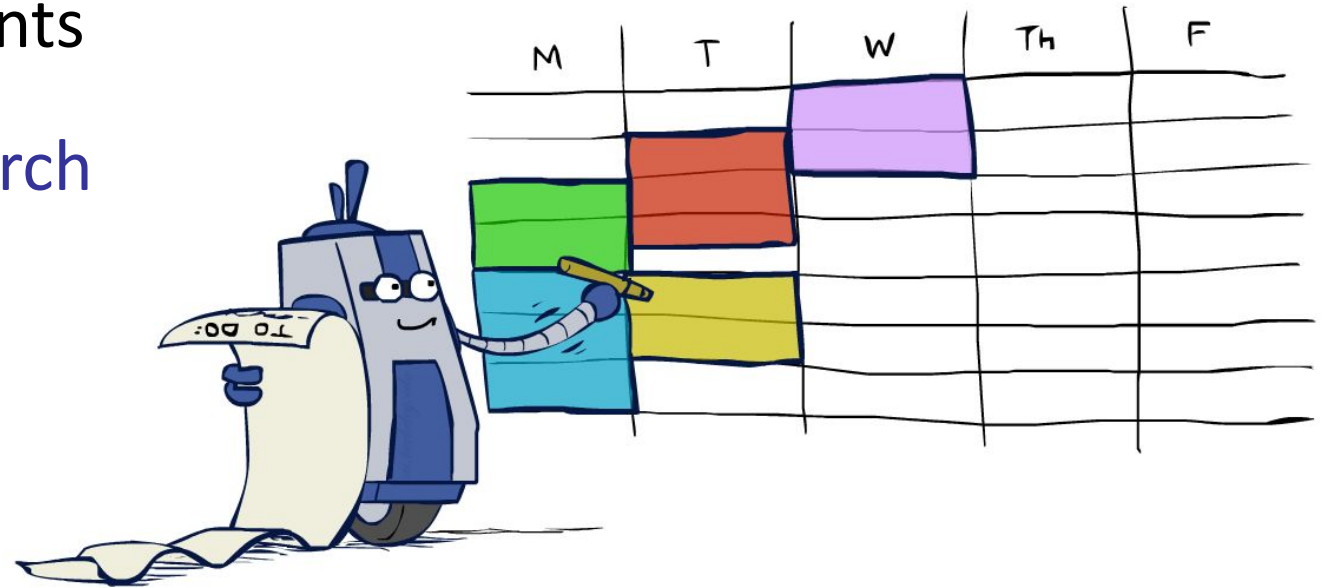
- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)!
- The same appears to be true for any randomly-generated CSP *except* in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



Summary: CSPs

- CSPs are a special kind of search problem:
 - States are partial assignments
 - Goal test defined by constraints
- Basic solution: backtracking search
- Speed-ups:
 - Ordering
 - Filtering
 - Structure
- Iterative min-conflicts is often effective in practice



Next Time: Adversarial Search!
