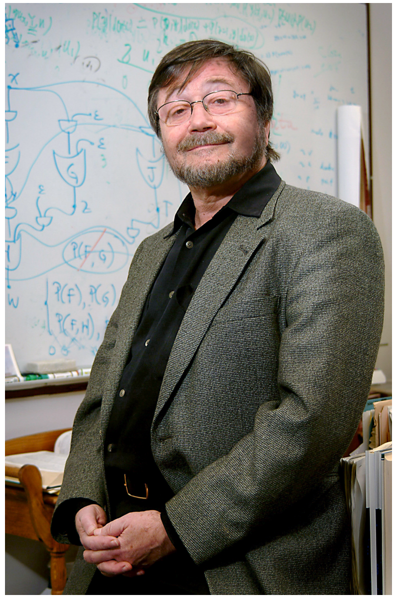


Artificial Intelligence

Constraint Satisfaction Problems (Part 2)

CS 444 – Spring 2021

Dr. Kevin Molloy

Department of Computer Science

James Madison University

JAMES MADISON UNIVERSITY

Much of this lecture is taken from
Dan Klein and Pieter Abbeel AI class at UC Berkeley

Today

- Review of A* Heuristics for PA 1
- Continue with Constraint Satisfaction problems
- Arc consistency AC-3 examples
- Problem Structure
- Min conflicts

Learning Objectives

- Apply the AC-3 to maintain arc consistency (MAC)
- Investigate the problem structure of CSPs for identify more efficient solutions using cutset conditioning and tree decomposition
- Apply min-conflicts algorithm and by able to code it to solve CSPs. Characterize the min-conflicts algorithm (runtime, completeness, etc).

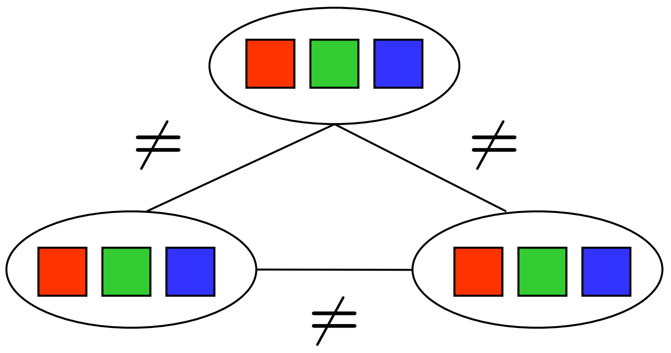
Figure from Berkley AI

Student Heuristic Presentation

- Alex Marasco – Finding/visit all the corners heuristics
- Garrett Christian -- Eat all the dots heuristic

CSP problems

- CSPs:
  - Variables
  - Domains
  - Constraints
    - Implicit (provide code to compute)
    - Explicit (provide a list of the legal tuples)
    - Types:
      - Unary (one variable)
      - Binary (two variables)
      - N-ary (n variables)



- Goals:
  - **In this class**: find any solution
  - Also: find all, find best, etc.

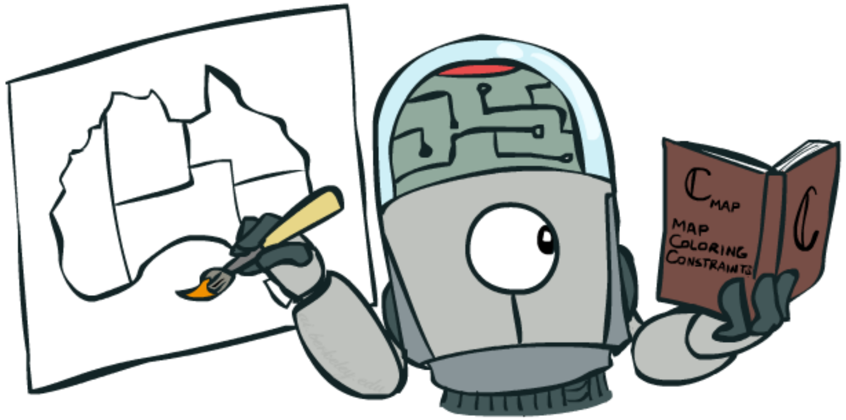


Figure from Berkley AI

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
```

Backtracking Example

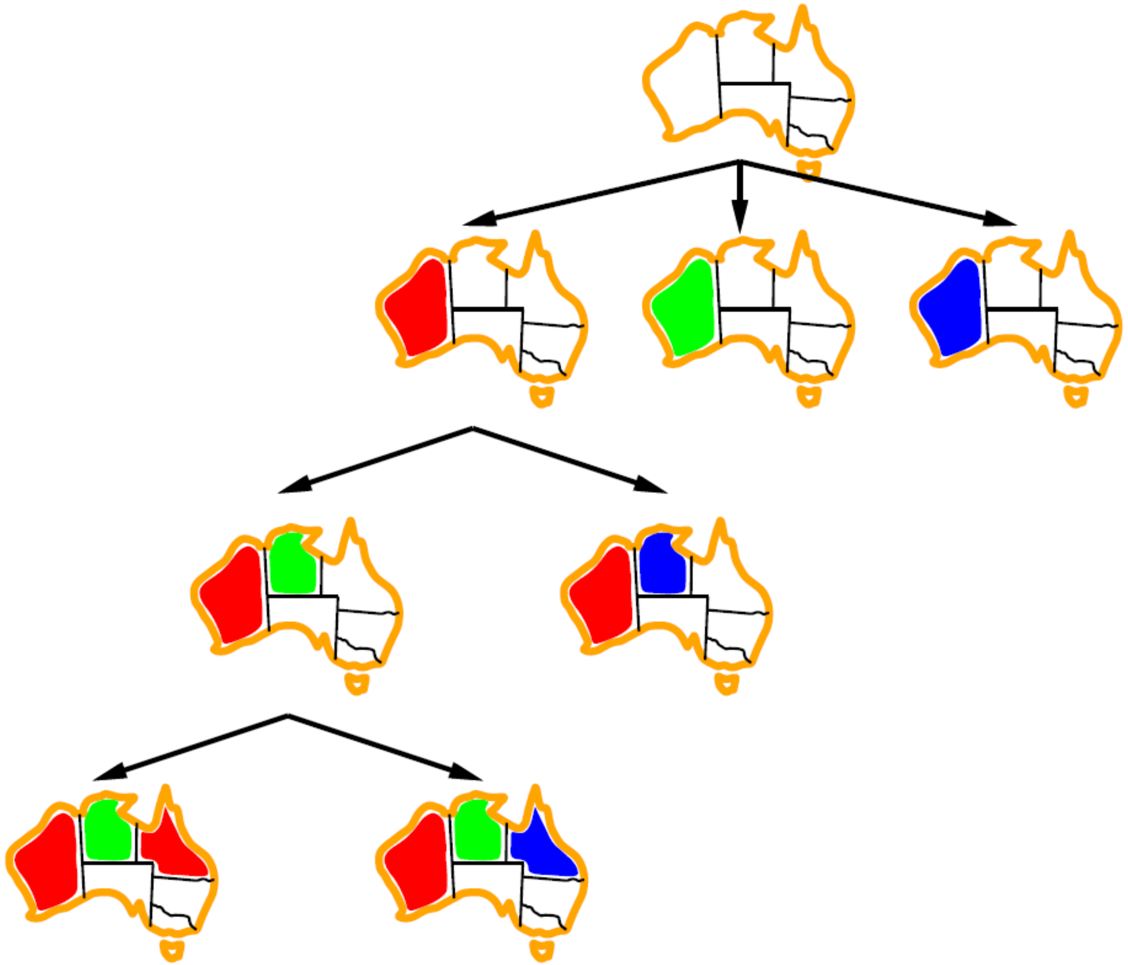


Figure from Berkley AI

Improving Backtracking

- General-purpose ideas give huge gains in speed
  - … but it's all still **NP-hard**

- **Filtering**: Can we detect inevitable failure early?

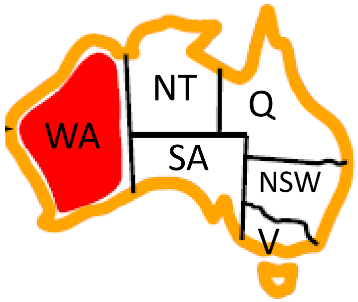
- **Ordering**:
  - Which **variable** should be assigned next? (MRV)
  - In what order should its **values** be tried? (LCV)

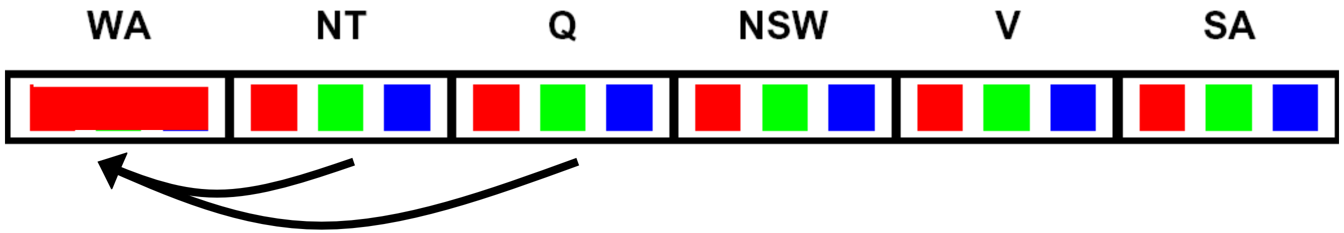
- **Structure**: Can we exploit the problem structure?

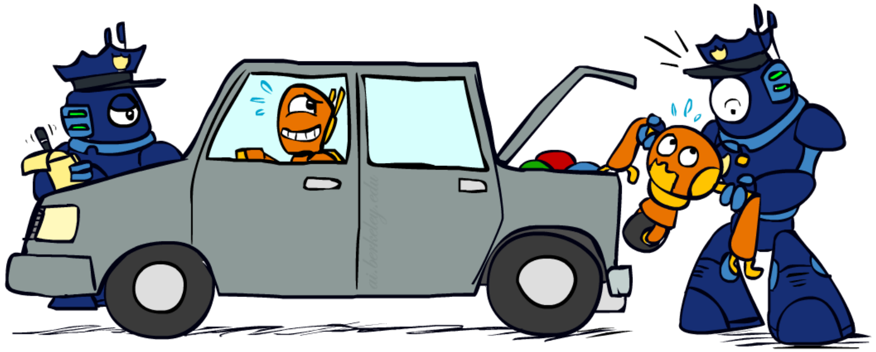
Figure from Berkley AI

Consistency of a Single Arc

- An arc X → 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





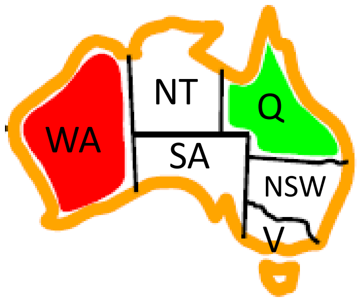


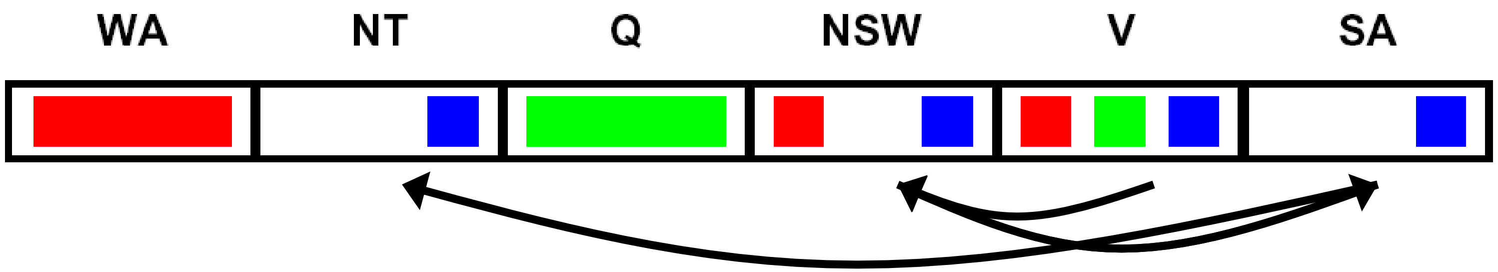
Delete from the tail!

- 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?

Remember: Delete from the tail!

Figure from Berkley AI

Enforcing Arc Consistency in a CSP

```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
         for each Xk in NEIGHBORS[Xi] do
            add (Xk, Xi) to queue
```

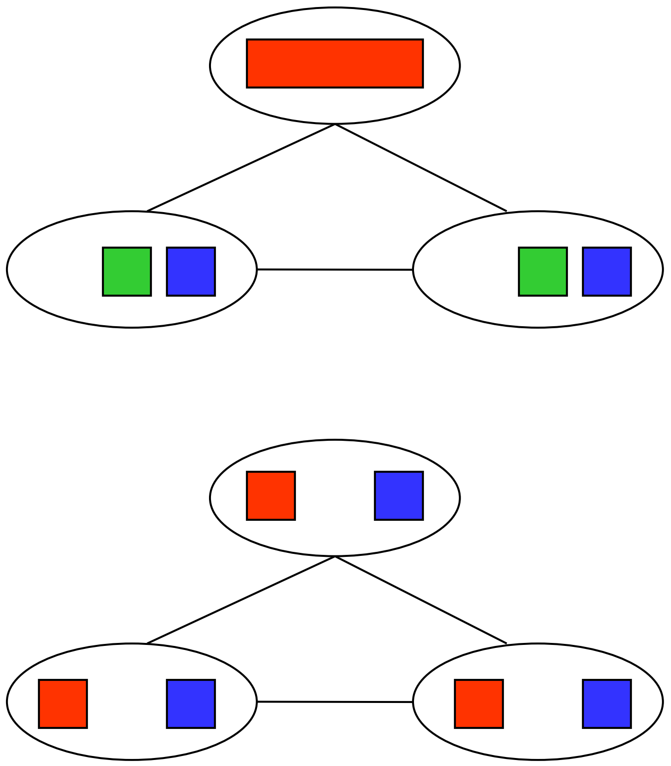
```
function REMOVE-INCONSISTENT-VALUES( Xi, Xj) returns true iff succeeds
   removed ← false
   for each x in DOMAIN[Xi] do
      if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
         then delete x from DOMAIN[Xi]; removed ← true
   return removed
```

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

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!



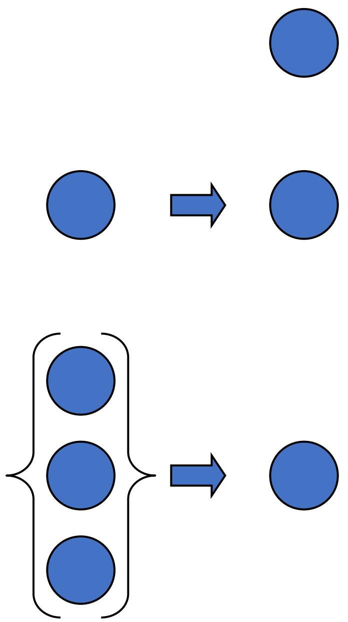
What went wrong here?

Figure from Berkley AI

JAMES MADISON UNIVERSITY

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 k^{th} node.



- Higher k more expensive to compute
- (You need to know the k=2 case: arc consistency)

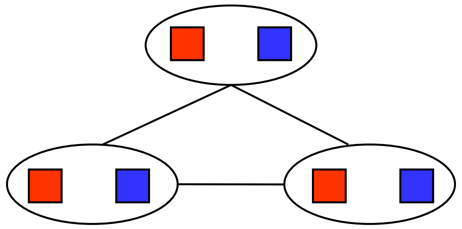


Figure from Berkley AI

Strong K-Consistency

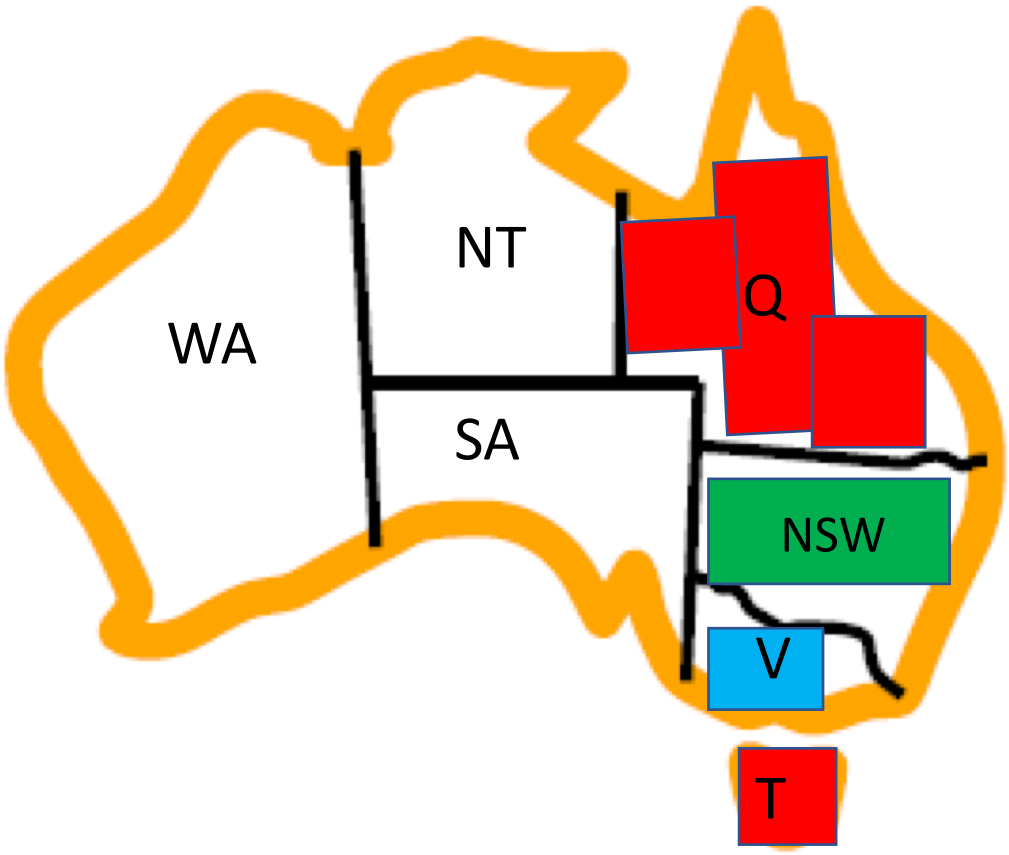
- Strong k-consistency: also k-1, k-2, … 1 consistent

- 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. k=3, called path consistency)

Intelligent Backtracking

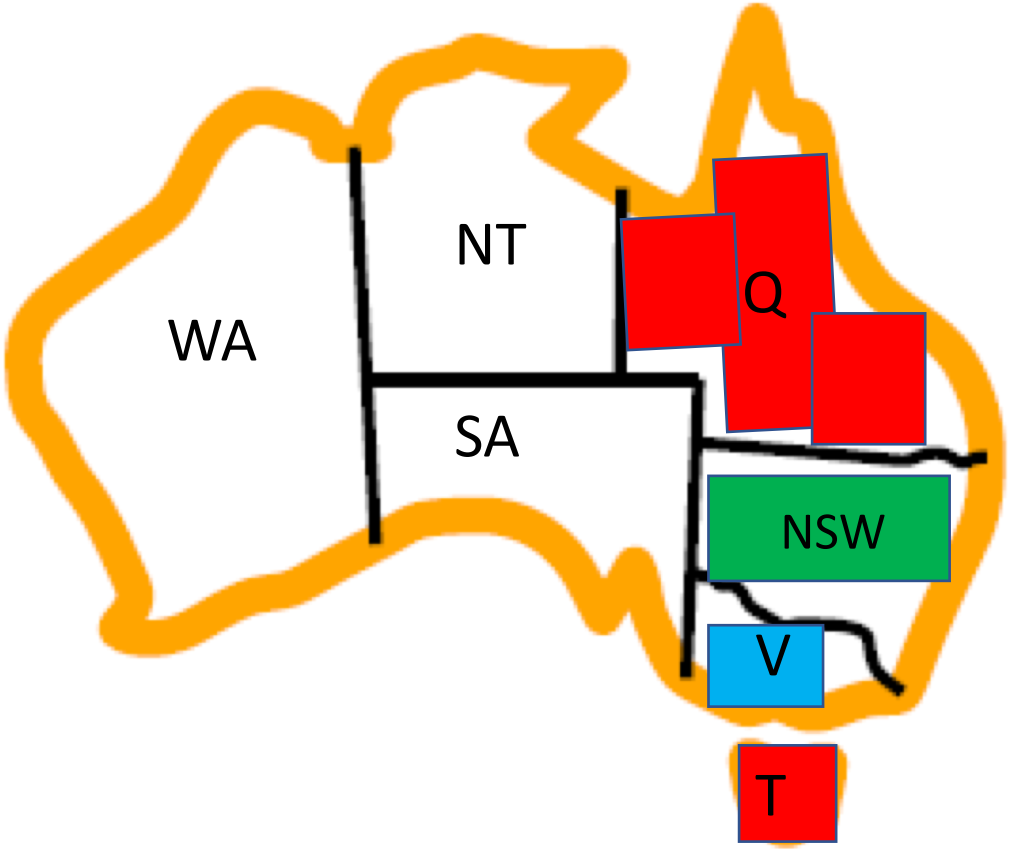


Variable assignment order: {Q, NSW, V, T, SA, WA, NT}

Partial Assignment: {Q = red, NSW=green, V=blue, T=red}

What does normal backtracking do when it tries to assignment **SA** a color?

Intelligent Backtracking



Variable assignment order: {Q, NSW, V, T, SA, WA, NT}

Partial Assignment: {Q = red, NSW=green, V=blue, T=red}

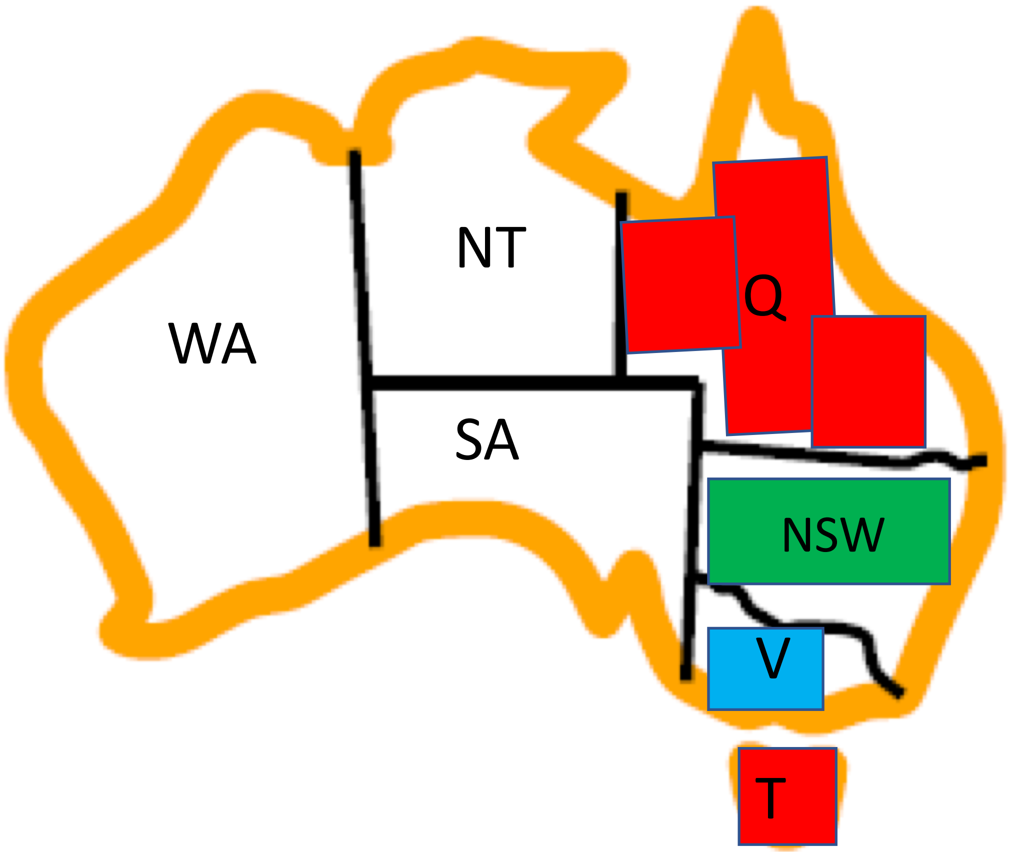
What does normal backtracking do when it tries to assignment **SA** a color?

- It tries all 3 colors. None of these work. So backtrack
- Change the color of T and try SA again
- Still no assignment works for SA. So backtrack.
- Etc.

How can we make this better?

Figure from Berkley AI

Intelligent Backtracking – Back jumping



Variable assignment order: {Q, NSW, V, T, SA, WA, NT}

Partial Assignment: {Q = red, NSW=green, V=blue, T=red}

Idea: Jump to a variable that is causing a problem.

Define a **conflict set**, which is built as we evaluate a variable. So, for SA, we check:

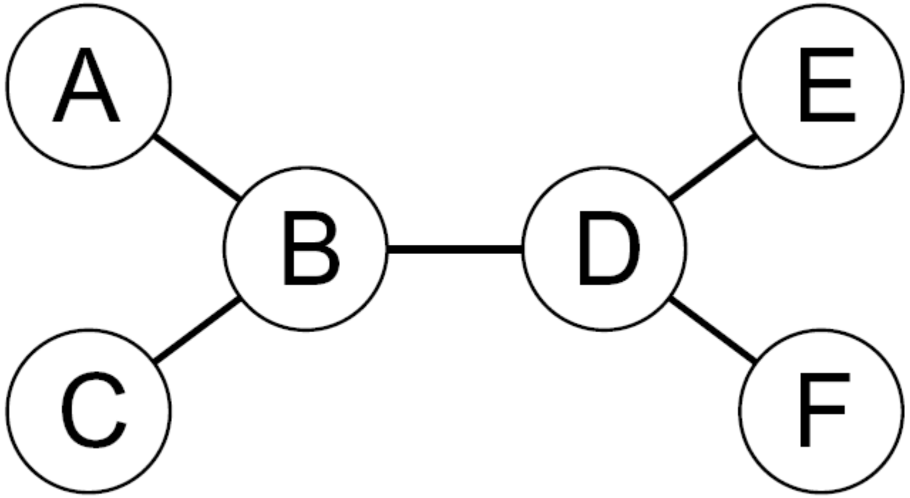
- Can't use red, Q is added to the conflict set for SA.
- Can't use green, NSW is added to the conflict set for SA.
- Can't use blue, V is added to the conflict set for SA.

Backtrack to at least one of these variables so we have a chance of correcting the issue.

https://inst.eecs.berkeley.edu/~cs188/fa19/assets/demos/csp/csp_demos.html

Figure from Berkley AI

Tree-Structured CSPs

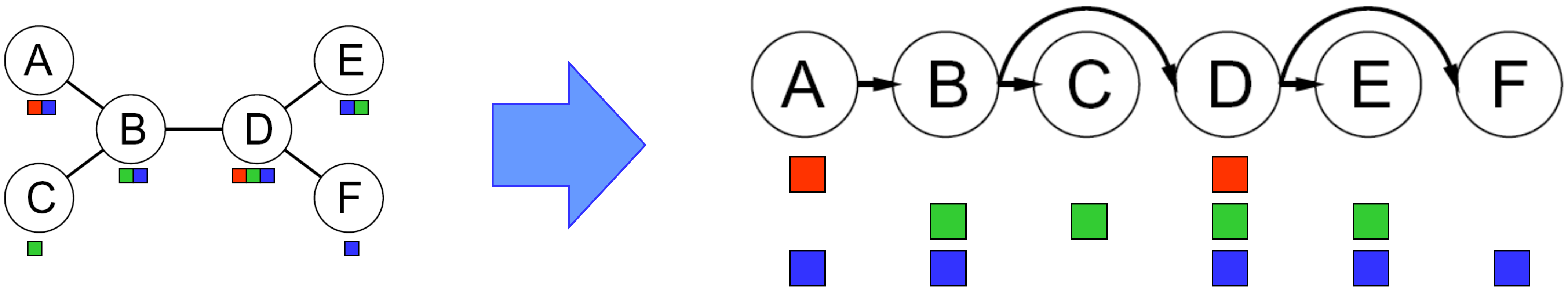


- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\ d^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$

- This property also applies to probabilistic reasoning (later): an example of the relation between syntactic restrictions and the complexity of reasoning

Tree-Structured CSPs

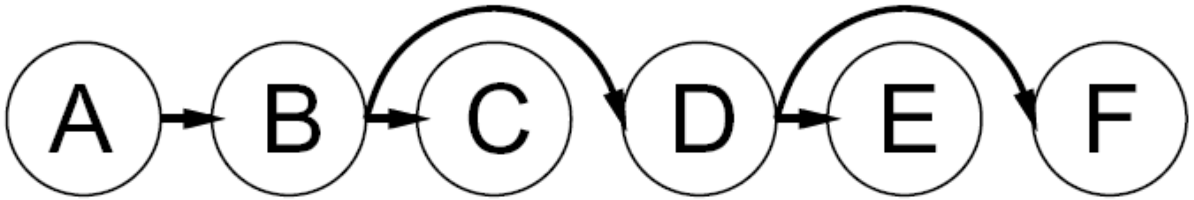
- Algorithm for tree-structured CSPs:
  - Order: Choose a root variable, order variables so that parents precede children



  - Remove backward: For i = n : 2, apply RemoveInconsistent(Parent(X_i),X_i)
  - Assign forward: For i = 1 : n, assign X_i consistently with 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→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)



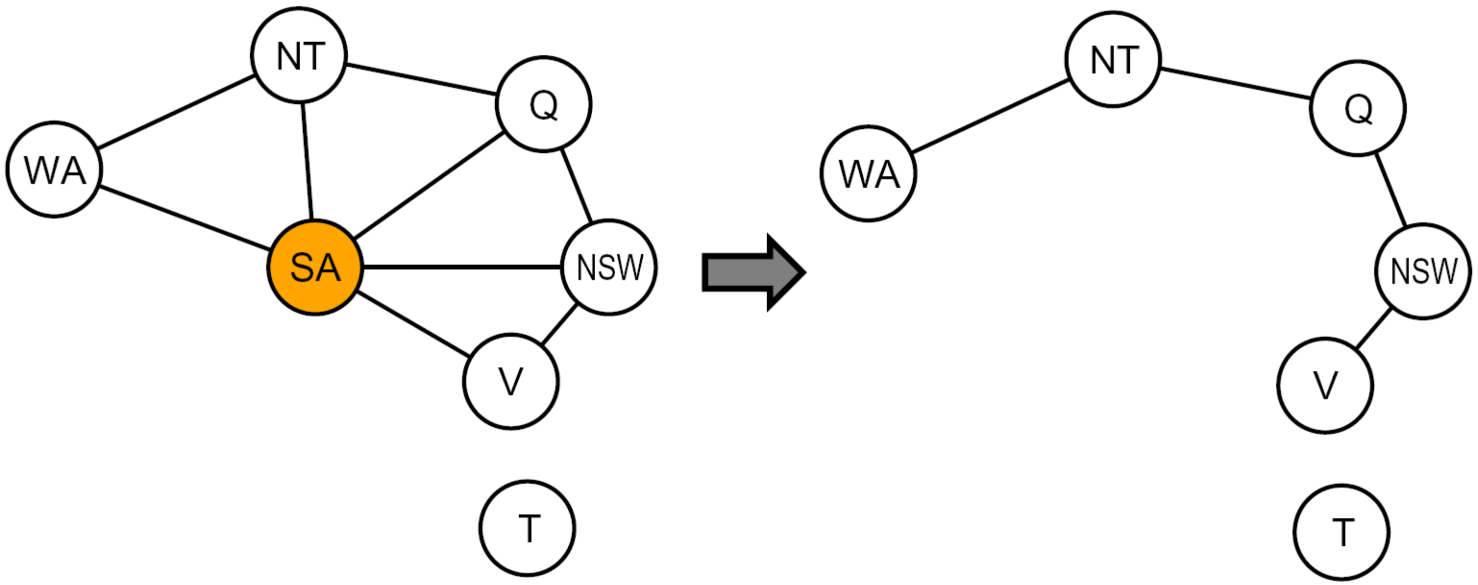
- Claim 2: If root-to-leaf arcs are consistent, forward assignment will not backtrack
- Proof: Induction on position

- Why doesn't this algorithm work with cycles in the constraint graph?

- Note: we'll see this basic idea again with Bayes' nets

Figure from Berkley AI

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

Figure from Berkley AI

Cutset Conditioning

Choose a cutset

Instantiate the cutset (all possible ways)

Compute residual CSP for each assignment

Solve the residual CSPs (tree structured)

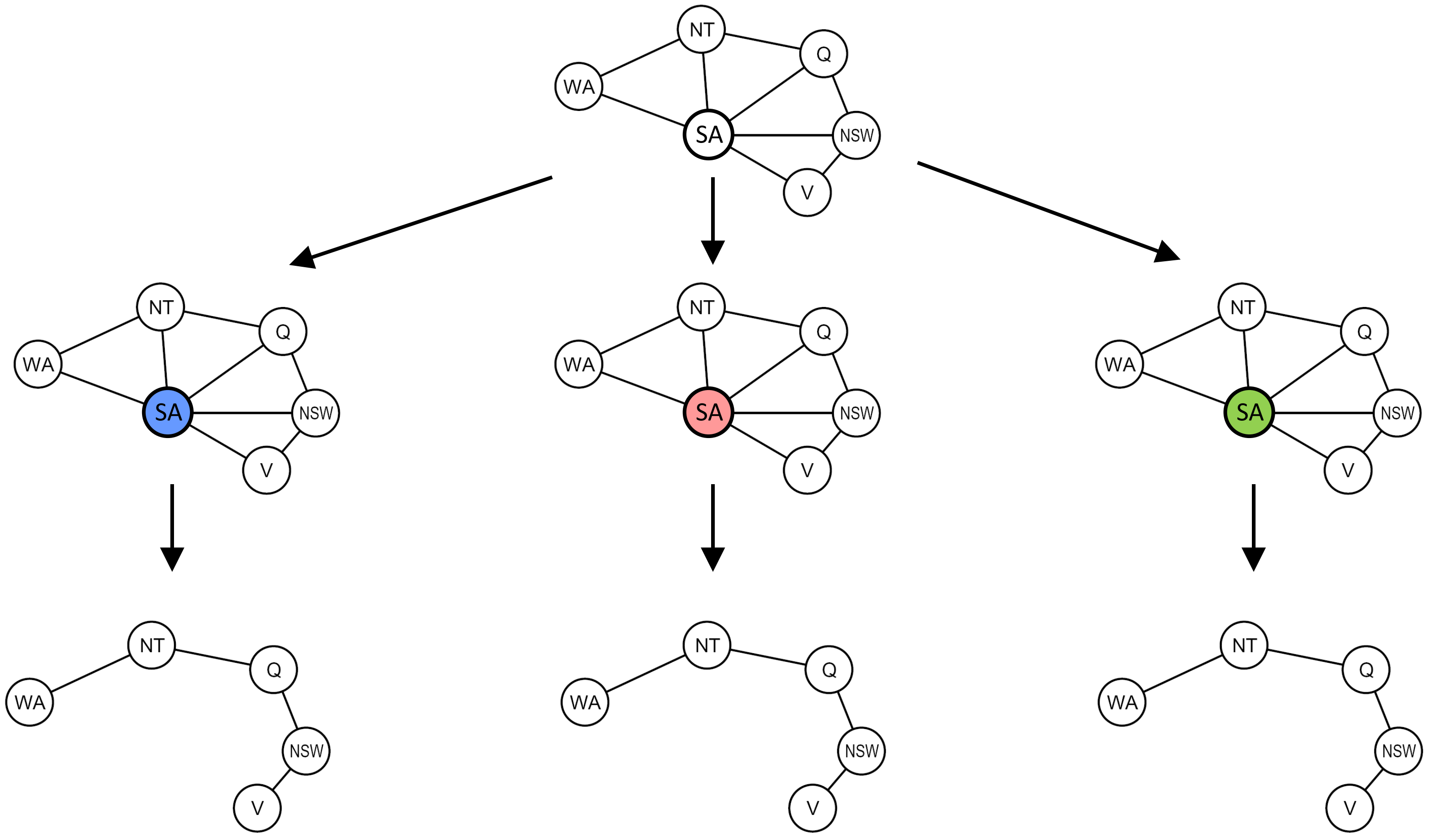


Figure from Berkley AI

Cutset Quiz

- Find the smallest cutset for the graph below.

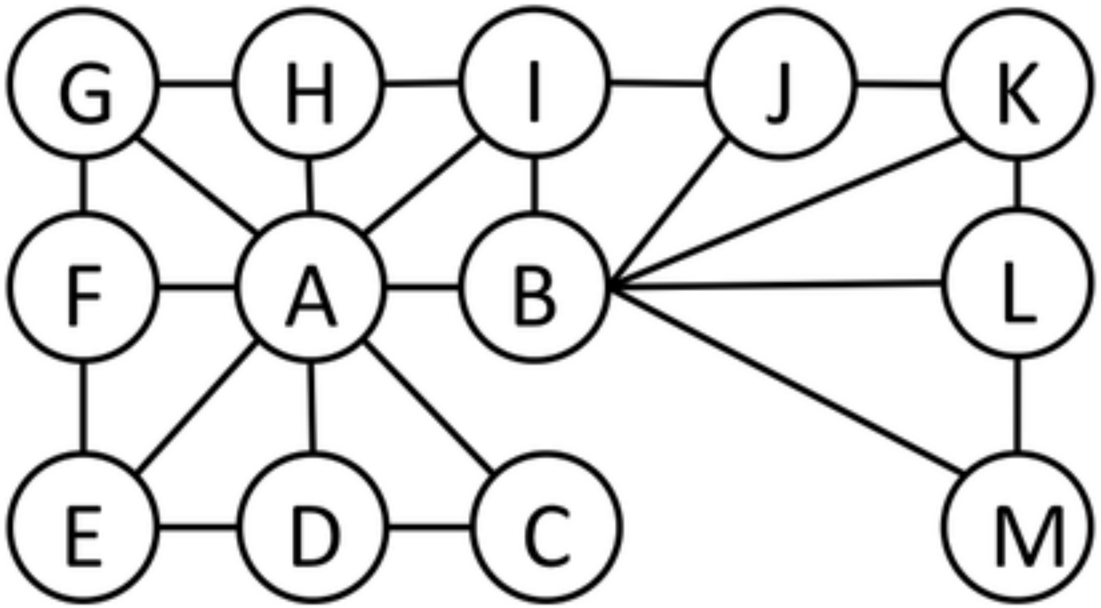
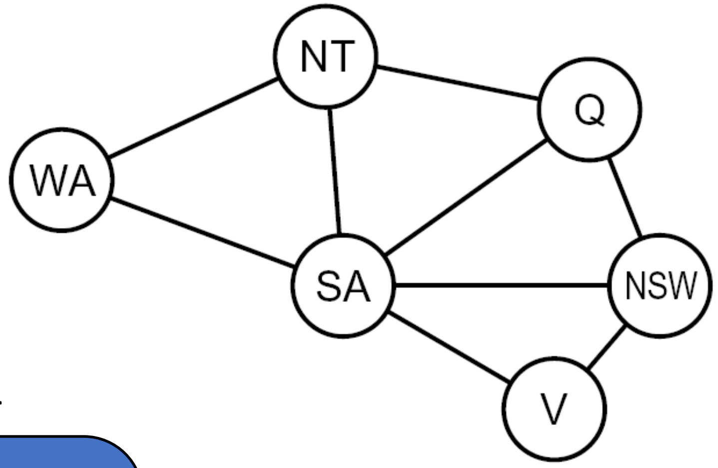
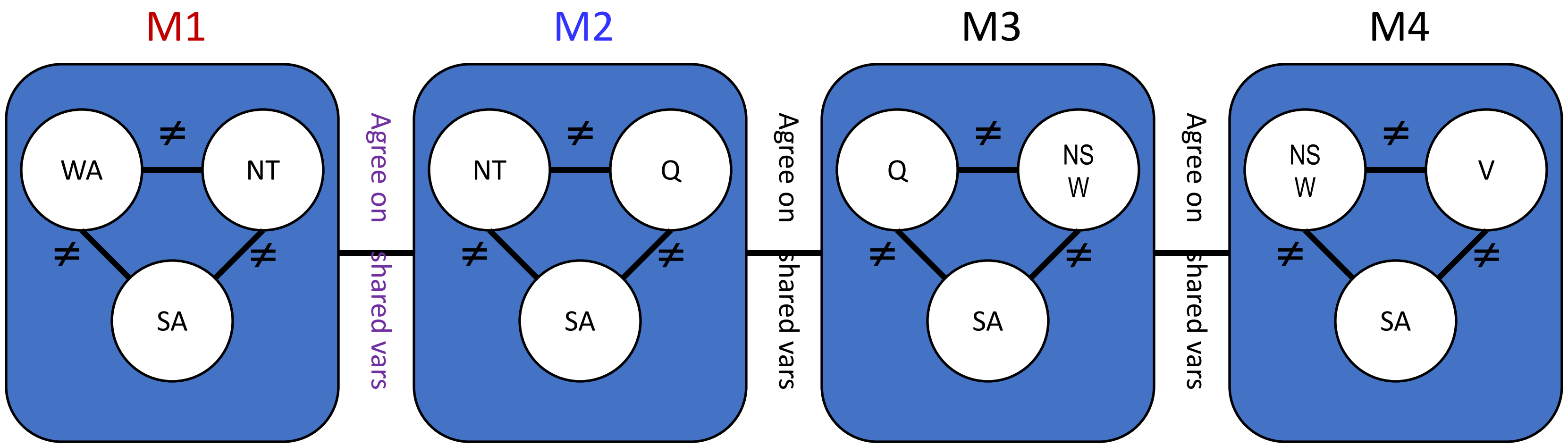


Figure from Berkley AI

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





{(WA=r,SA=g,NT=b),
(WA=b,SA=r,NT=g),
…}

{(NT=r,SA=g,Q=b),
(NT=b,SA=g,Q=r),
…}

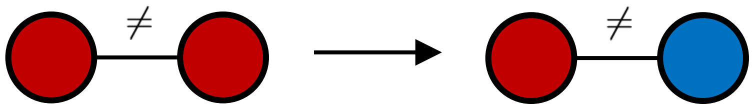
Agree: (M1,M2) ∈
{((WA=g,SA=g,NT=g), (NT=g,SA=g,Q=g)), …}

Figure from Berkley AI

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 heuristic:
    Choose a value that violates the fewest constraints
    I.e., hill climb with h(n) = total number of violated constraints

JAMES MADISON UNIVERSITY

Figure from Berkley AI

Performance of Different CSP Algorithms

Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV	Min-Conflicts
USA (4 color)	(> 1,000,000)				
n-Queens	(> 40,000,000)				

Figure from Berkley AI

Performance of Different CSP Algorithms

Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV	Min-Conflicts
USA (4 color)	(> 1,000,000)	(> 1,000,000)			
n-Queens	(> 40,000,000)	13,500,000			

Figure from Berkley AI

Performance of Different CSP Algorithms

Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV	Min-Conflicts
USA (4 color)	(> 1,000,000)	(> 1,000,000)	2,000		
n-Queens	(> 40,000,000)	13,500,000	(> 40,000,000)		

Figure from Berkley AI

Performance of Different CSP Algorithms

Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV	Min-Conflicts
USA (4 color)	(> 1,000,000)	(> 1,000,000)	2,000	60	
n-Queens	(> 40,000,000)	13,500,000	(> 40,000,000)	817,000	

Figure from Berkley AI

Performance of Different CSP Algorithms

Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV	Min-Conflicts
USA (4 color)	(> 1,000,000)	(> 1,000,000)	2,000	60	64
n-Queens	(> 40,000,000)	13,500,000	(> 40,000,000)	817,000	4,000

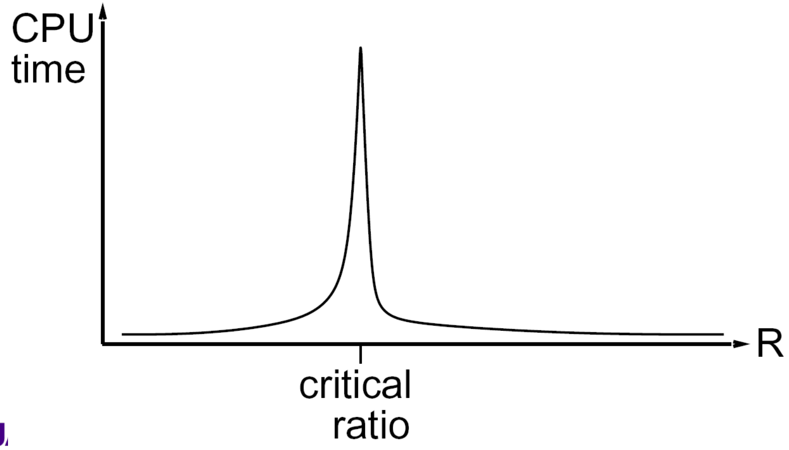
Figure from Berkley AI

Don't Make Things too Complicated

- 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

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



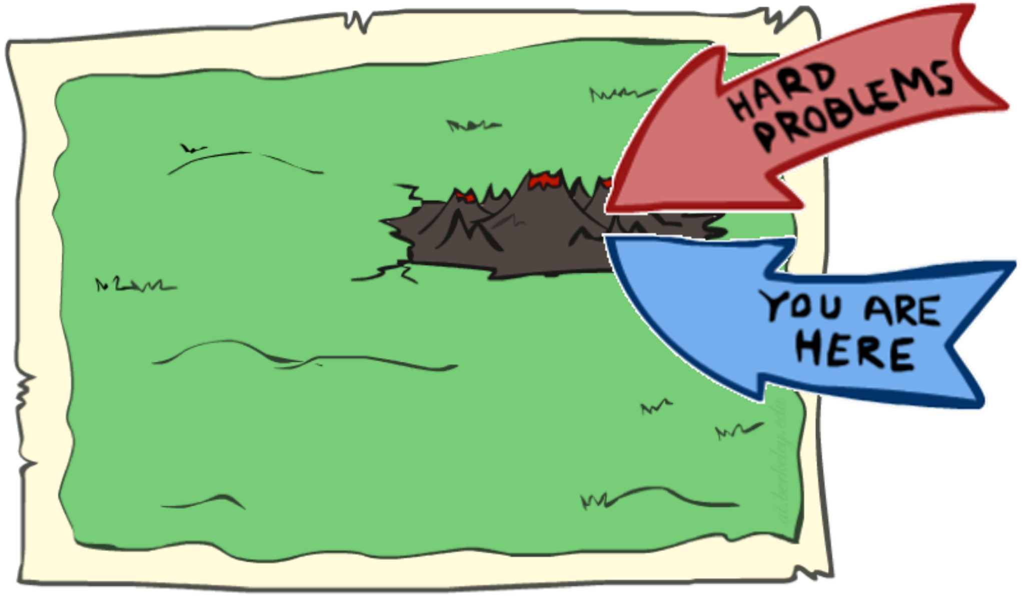


Figure from Berkley AI

CSP Summary

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

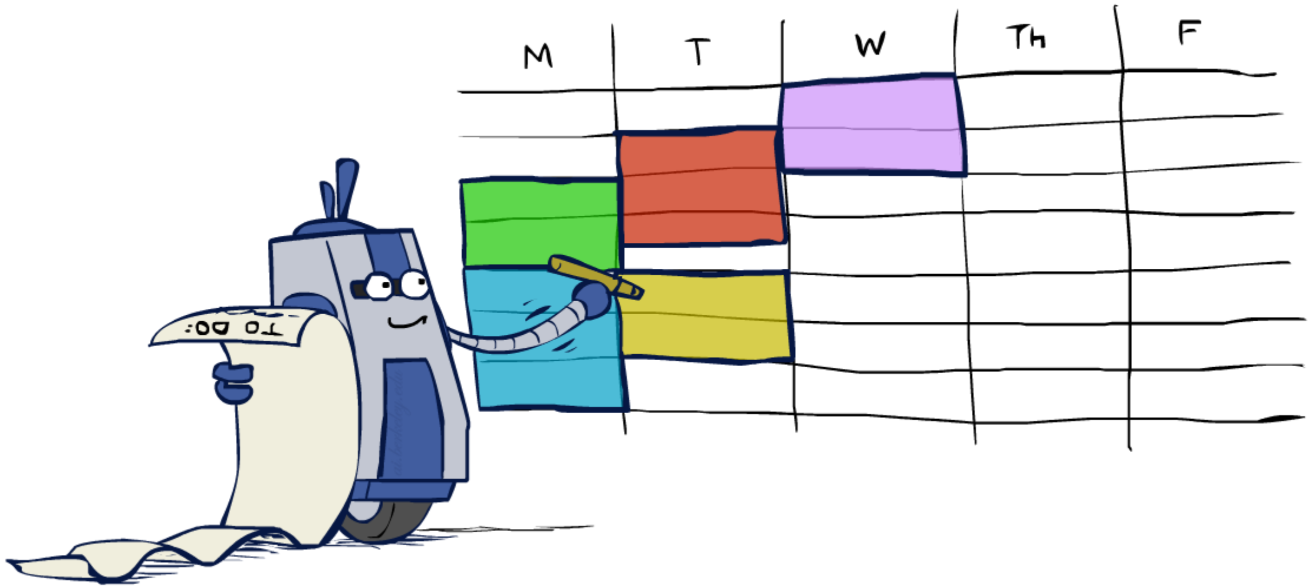


Figure from Berkley AI