Announcement

- TA office hour
  - Wed evening: 20:00-21:00
  - Location: SIST 1B-101
- Programming Assignment 1:
  - Released on Thursday
- Autolab registration
  - See email or BB announcement

Constraint Satisfaction Problems

AIMA Chapter 6

[Adapted from slides by Dan Klein and Pieter Abbeel at UC Berkeley (ai.berkeley.edu).]

What is Search For?

- Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space

Type A

- Planning: (optimal) sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics give problem-specific guidance

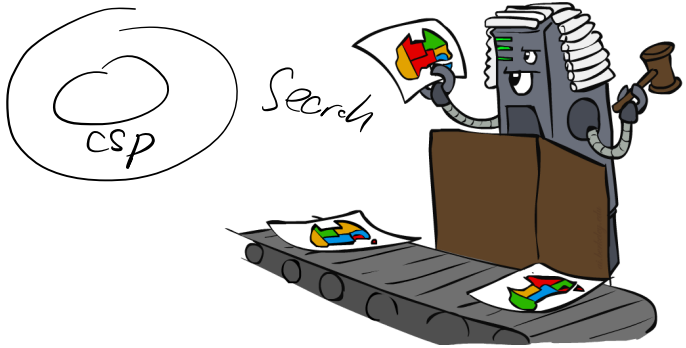
B

- Identification : assignments to variable
  - The goal itself is important, not the path
  - All paths may be at the same depth (for some formulations)

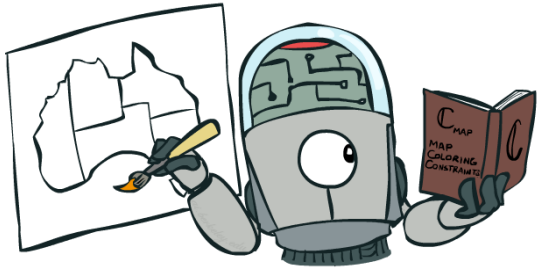
goal

Search Problems

- Standard search problems:
  - State is a "black box": arbitrary data structure
  - Goal test can be any function over states
  - Successor function can also be anything

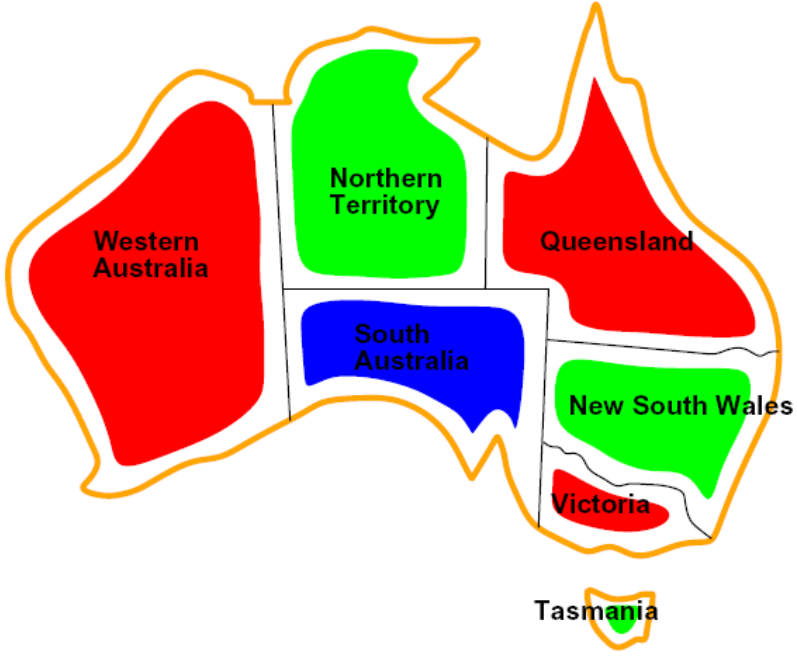


- Constraint satisfaction problems (CSPs):
  - A special subset of search problems
  - State is defined by variables X_i with values from a domain D (sometimes D depends on i)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables
- CSPs are specialized for identification problems



Xi domain goal test
state D
relation

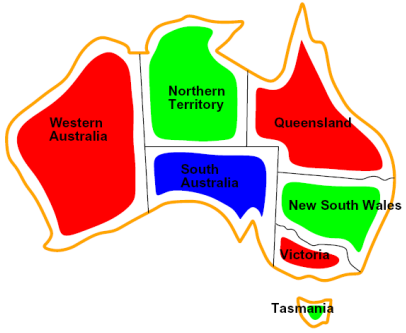
CSP Examples

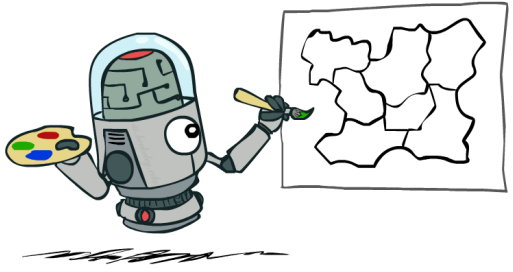


Example: Map Coloring

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

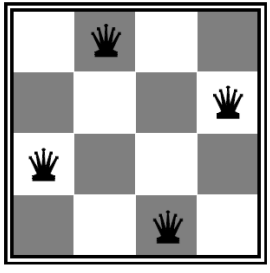
  {WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}

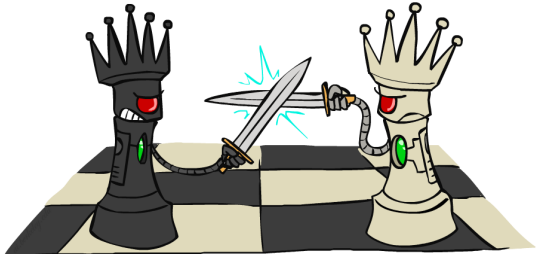




Example: N-Queens

- Formulation 1:
  - Variables: X_{ij}
  - Domains: $\{0, 1\}$
  - Constraints



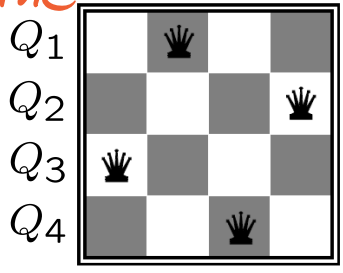


$$\forall i, j, k \quad (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$

$$\sum_{i,j} X_{ij} = N$$

Example: N-Queens

- Formulation 2: assume 1 queen in every Line
  - Variables: Q_k
  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:



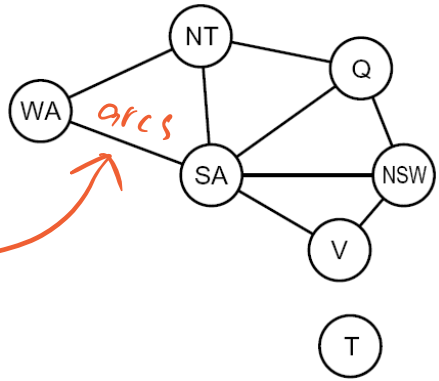
Implicit: $\forall i, j$ non-threatening(Q_i, Q_j)

Explicit: $(Q_1, Q_2) \in \{(1,3), (1,4), \ldots\}$

$\ldots$

Constraint Graphs

- Binary CSP: each constraint relates (at most) two variables
- Binary constraint graph: nodes are variables, arcs show constraints
- Now we can develop general-purpose CSP algorithms on the constraint graph
- What if there are constraints relating more than two variables?



Q why eg. color?

Use graph to speed up search and find cases like isolated nodes.

Example: Cryptarithmetic

- Variables:

  $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$

  Carries

- Domains:

  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

- Constraints:

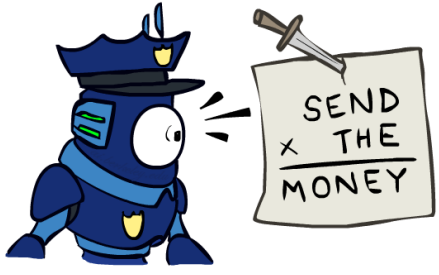
  alldiff(F, T, U, W, R, O)

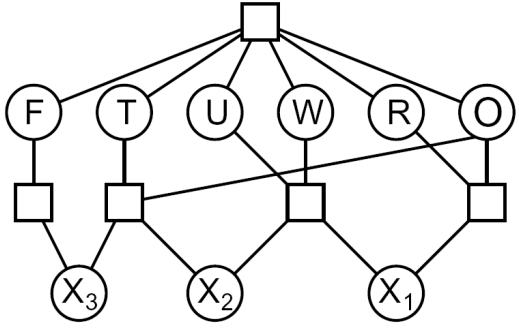
  $O + O = R + 10 \cdot X_1$

  $\cdots$

X_1

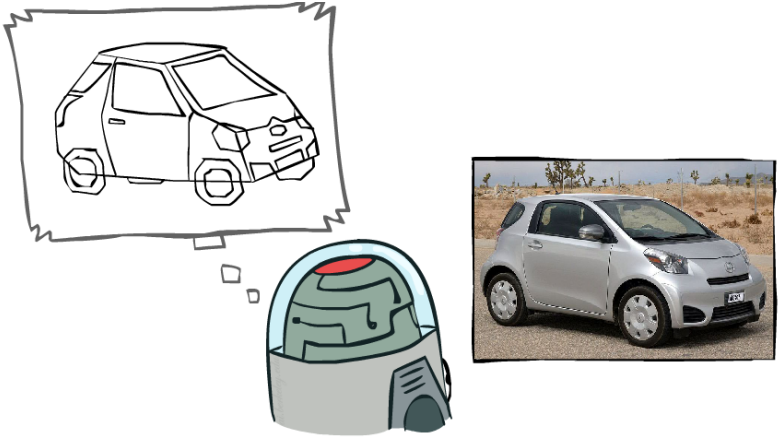
$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$

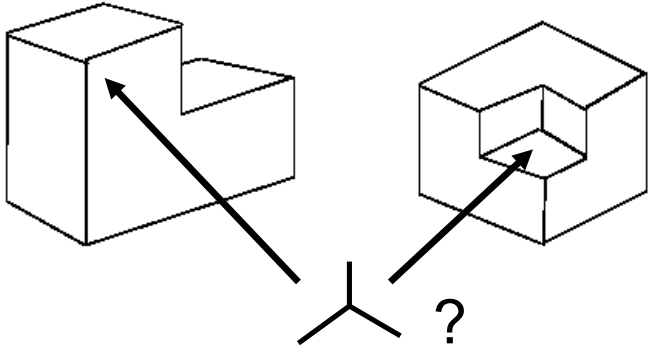




Example: The Waltz Algorithm

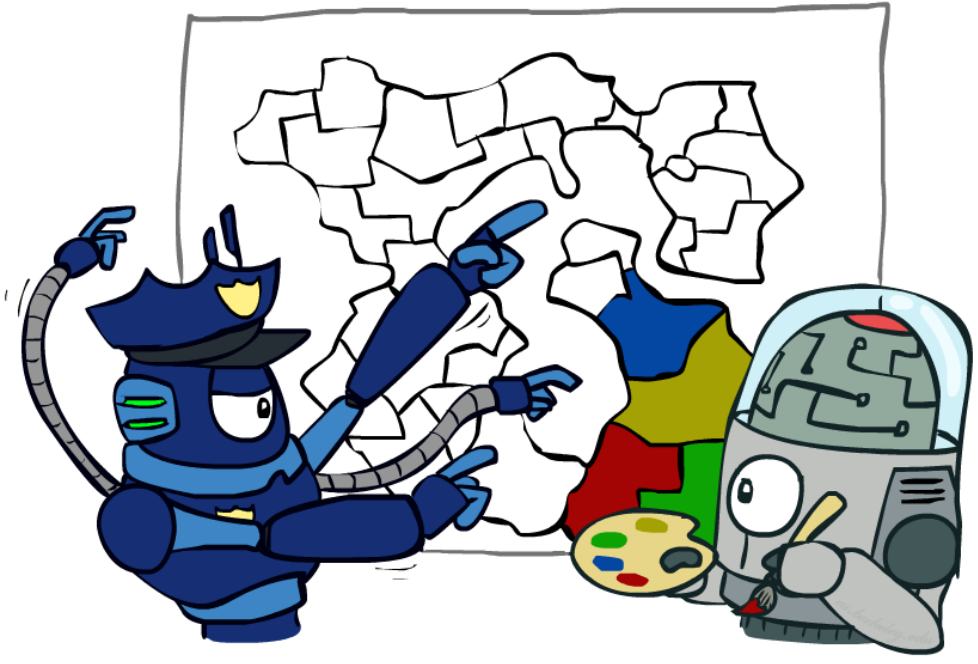
- The Waltz algorithm is for interpreting line drawings of solid polyhedra as 3D objects
- An early example of an AI computation posed as a CSP





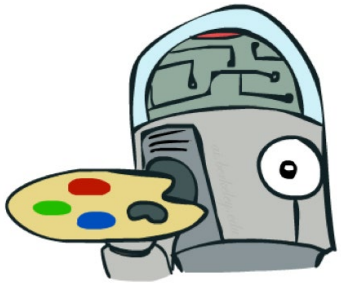
- Approach:
  - Each intersection is a variable
  - Adjacent intersections impose constraints on each other
  - Solutions are physically realizable 3D interpretations

Varieties of CSPs and Constraints



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 dates for each job
    - Linear constraints solvable, nonlinear undecidable



- Continuous variables
  - E.g., start/end times for Hubble Telescope observations
  - Linear constraints solvable in polynomial time by LP methods

Varieties of Constraints

- Varieties of Constraints
  - Unary constraints involve a single variable (equivalent to reducing domains), e.g.:

    $$SA \neq green$$

  - Binary constraints involve pairs of variables, e.g.:

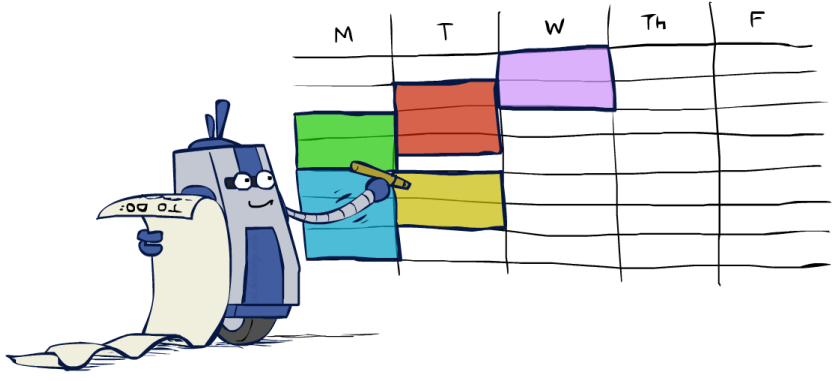
    $$SA \neq WA$$

  - Higher-order constraints involve 3 or more variables:

- 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
- Circuit layout
- Fault diagnosis
- … lots more!



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

Solving CSPs

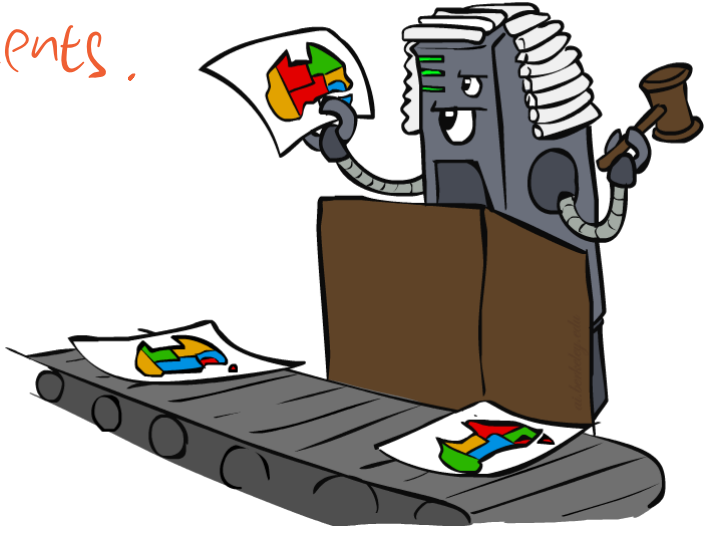
Standard Search Formulation

- Standard search formulation of CSPs
- States defined by the values assigned so far (partial assignments)
  - 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

Partial Assignments.

① { }

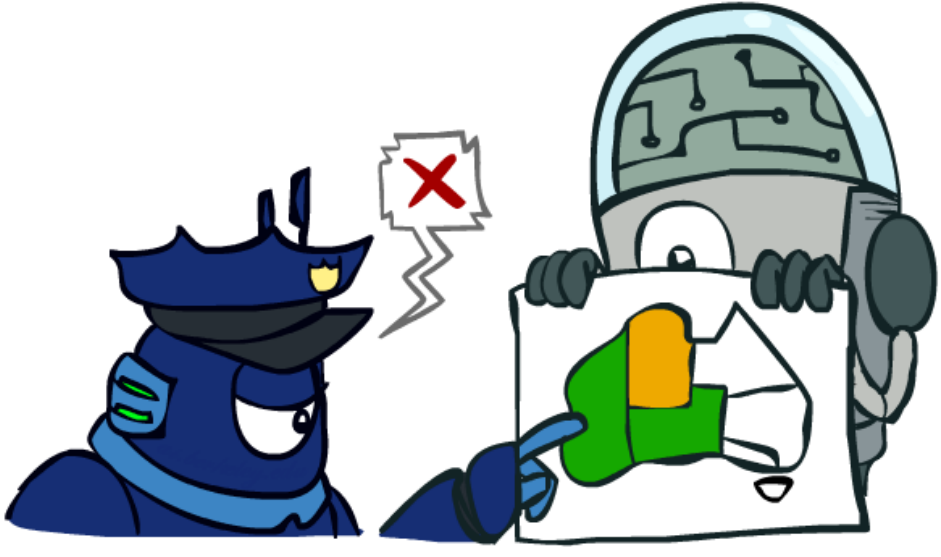
② {A=a . . . }



Search Methods

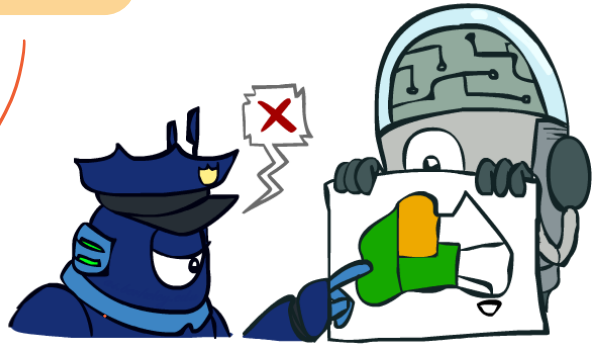
- What would DFS/BFS do?
  - Demo
  - What's wrong?

Backtracking Search

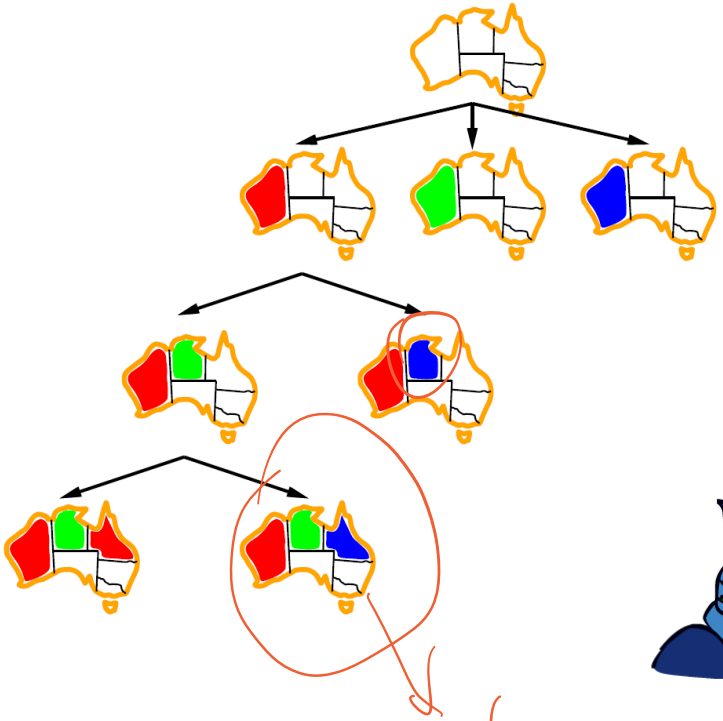


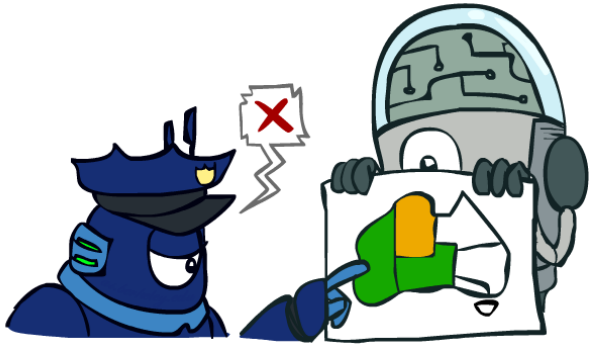
Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs
- Idea 1: One variable at a time
  - 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
- Idea 2: Check constraints as you go
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to check the constraints
  - "Incremental goal test"
- Depth-first search with these two improvements is called *backtracking search* (not the best name)
- Can solve n-queens for n ≈ 25



Backtracking Example





Backtracking Search

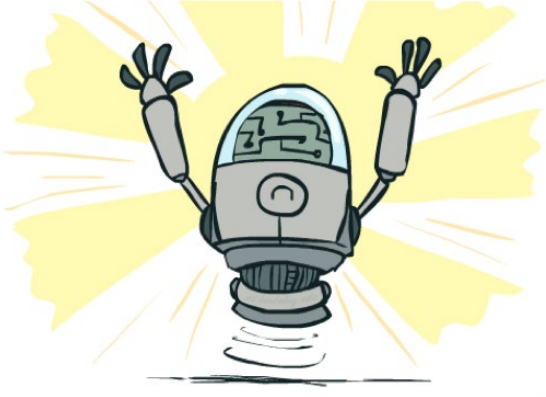
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

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

objects value, object (MRV, LCV ...)

Improving Backtracking

- General-purpose ideas give huge gains in speed
- Filtering: Can we detect inevitable failure early?
- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Structure: Can we exploit the problem structure?

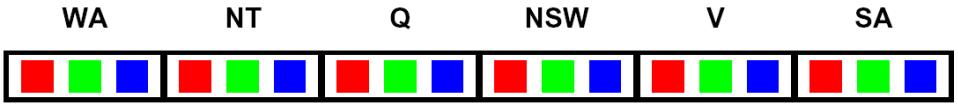


Filtering

filter out the nodes not considered

Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment; whenever any variable has no value left, we backtrack

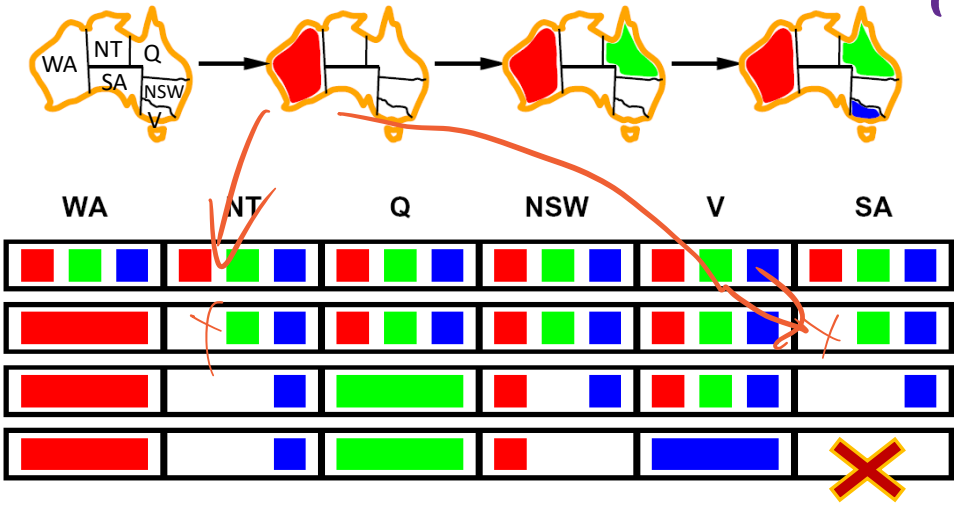


Forward: assigned value.

look into if contrdiction in value.

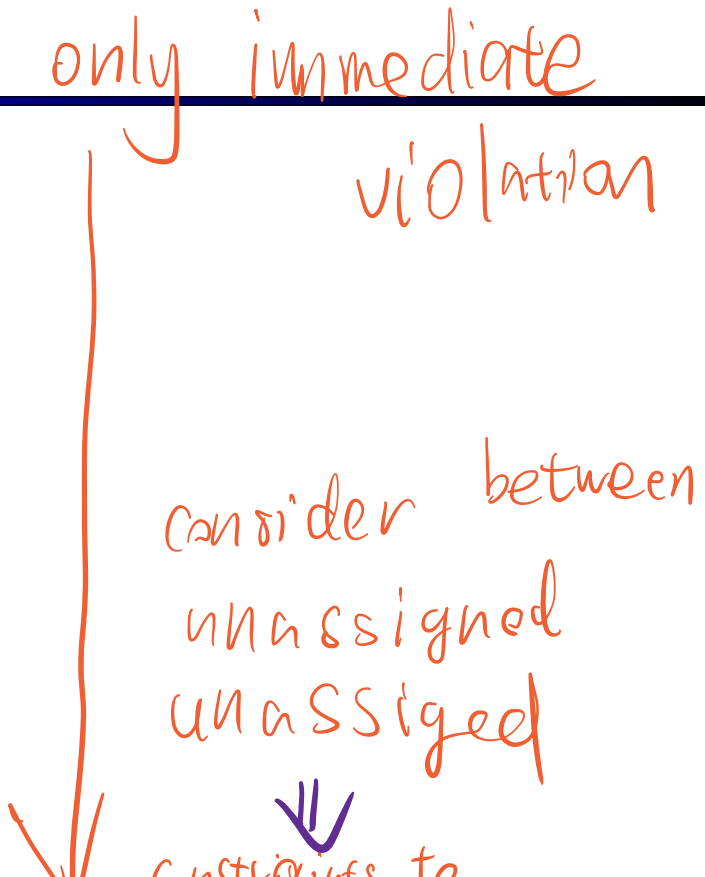
Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment; whenever any variable has no value left, we backtrack



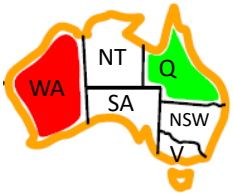
Demo

- Backtracking
- Backtracking with Forward Checking

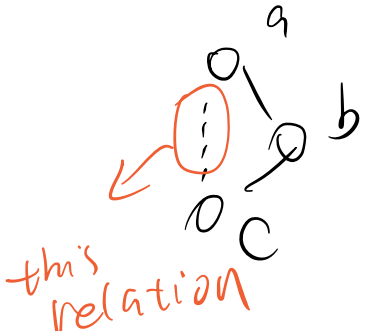


Filtering: Constraint Propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

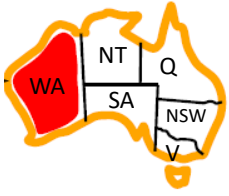


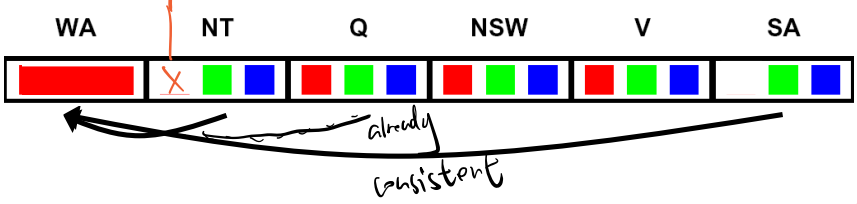
- NT and SA cannot both be blue!
- Can we detect this early?

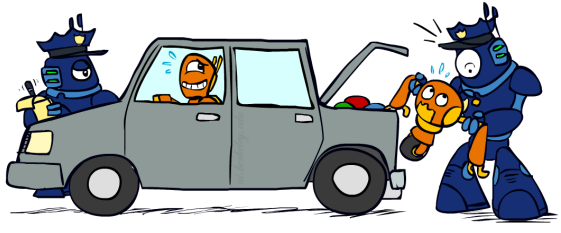


Consistency of A Single Arc

- 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







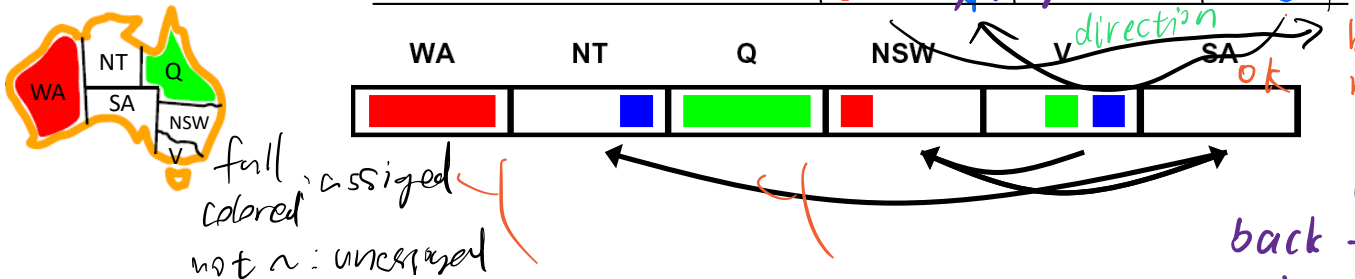
Delete from the tail!

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 Y loses a value, then arc X → Y needs to be rechecked!
- Arc consistency detects failure earlier than forward checking
- What's the downside of enforcing arc consistency?

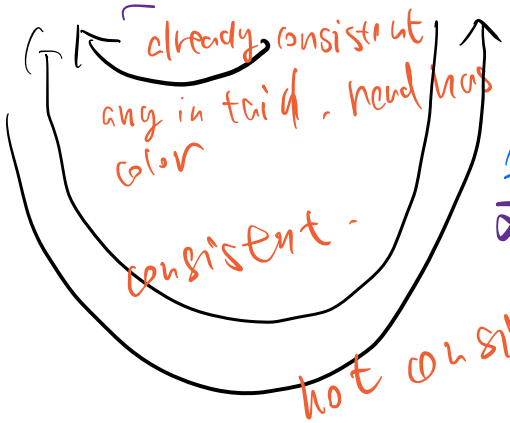
Remember: Delete from the tail!

3. NSW should be checked!

judge

WA	NT	Q	NSW 2	V 4	SA
R	b	G	r ~~b~~	~~rgb~~	b

(forwarded)



1

2 → 6 b!

1. delete from tail

2. tail → head

3. head changed

recheck head ←

all

2 → delete b.

4. for any color in tail

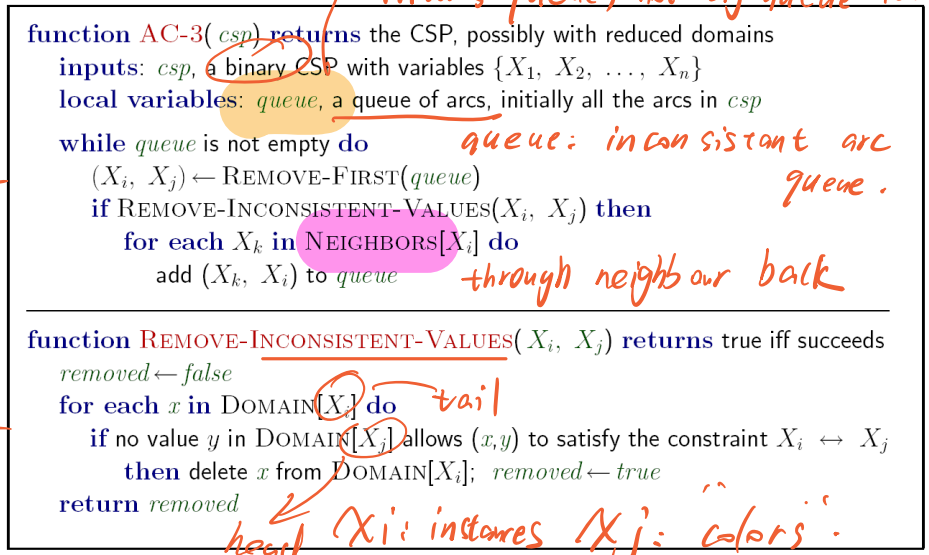
exist sth in head consist

5. when fail back track

to last time we choose

a value

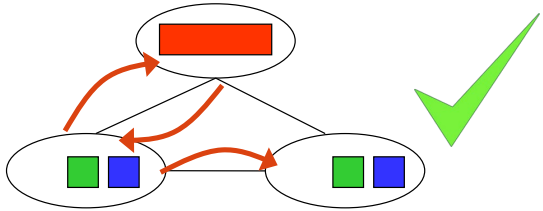
Enforcing Arc Consistency in a CSP

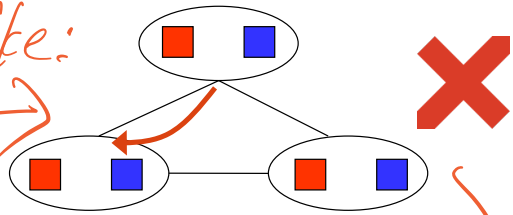


- Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$

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)



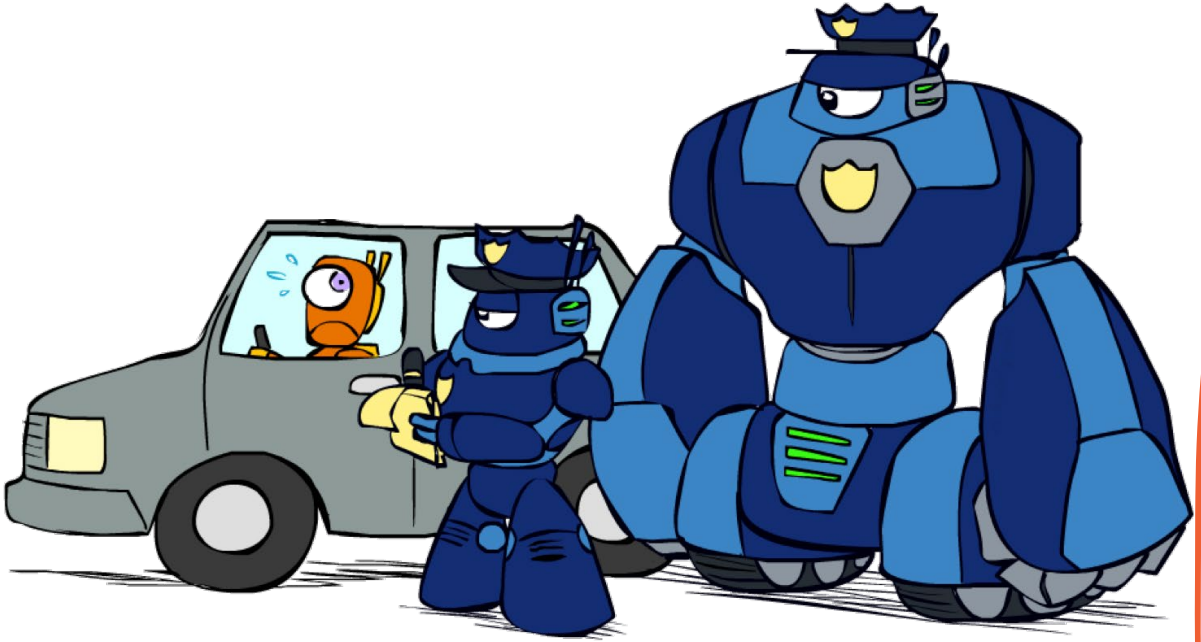


- Arc consistency still runs inside a backtracking search!

Demo

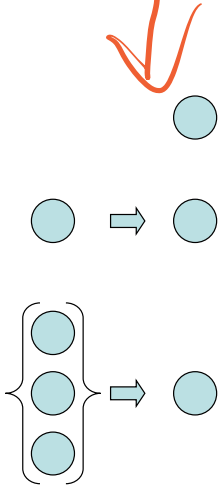
- Backtracking with Forward Checking
- Backtracking with Arc Consistency

K-Consistency



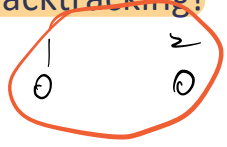
K-Consistency

- 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



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)



NP Hard

assigned with

if can not find val consistent => fail

Ordering

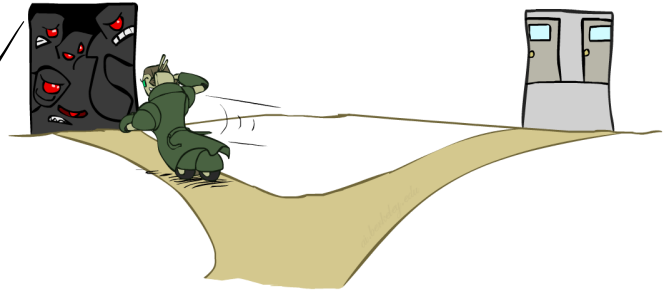
MRV Ordering: Minimum Remaining Values

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

Pick

3 color
2 color
1 color ✓

"Fail Fast" easy to correct



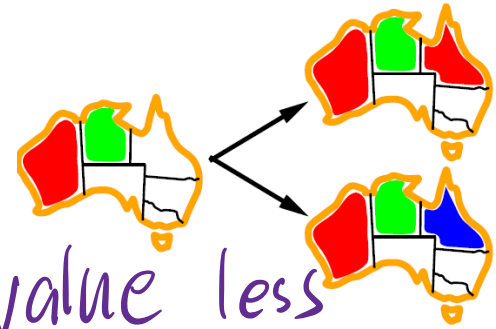
Choose variable

LCV

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)

排除

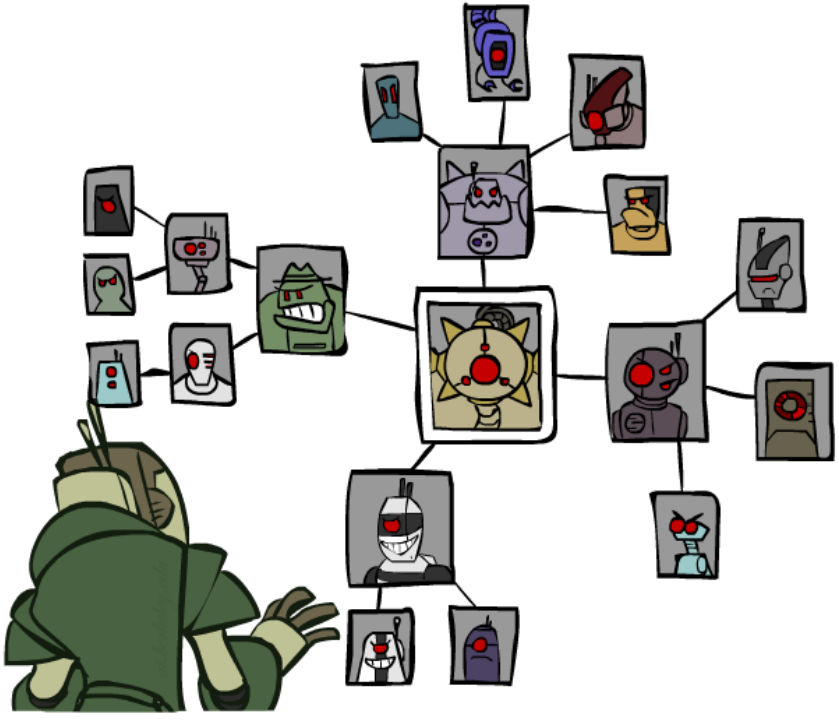


- Combining these ordering ideas makes 1000 queens feasible

Given variable,

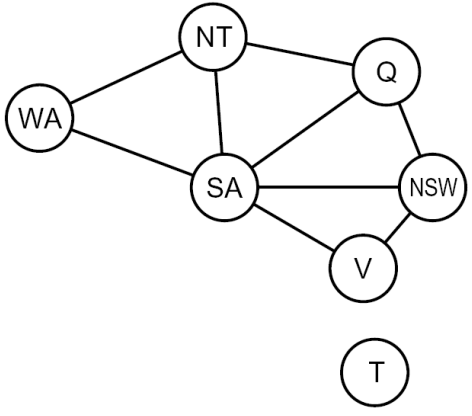
Choose Value

Structure

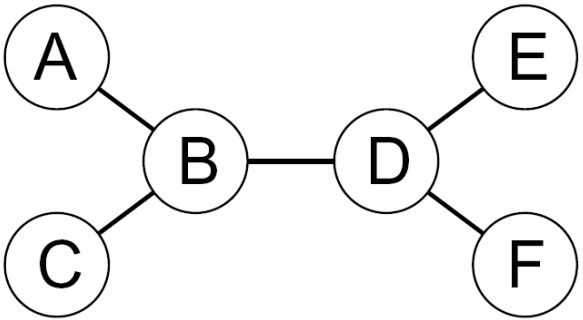


Problem Structure

- 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



Tree-Structured CSPs



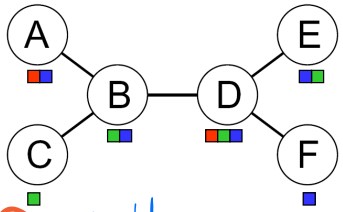
(linear) Time.

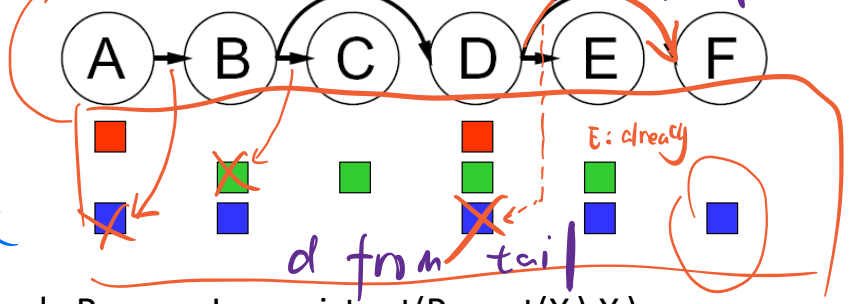
- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n d^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$
- This property also applies to probabilistic reasoning (later)
- An example of the relation between syntactic restrictions and the complexity of reasoning

make it a tree!

Tree-Structured CSPs

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





Still may be back track

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

- Runtime: $O(n\ d^2)$

when do this will consistent

assign values

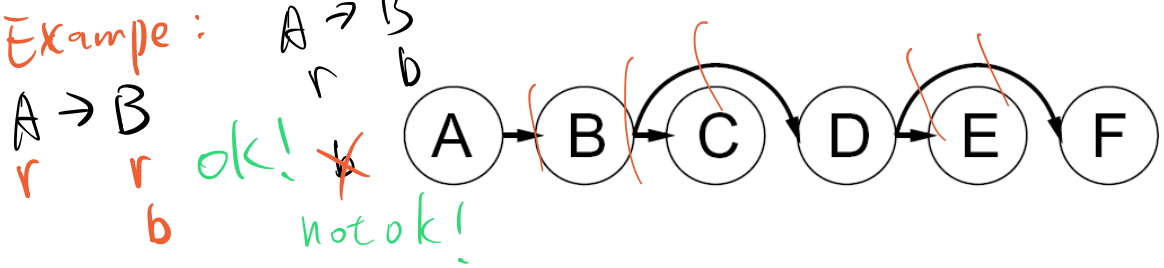
A B C D E F

B B G G B B

back: for each node make arc pointing to it consistent

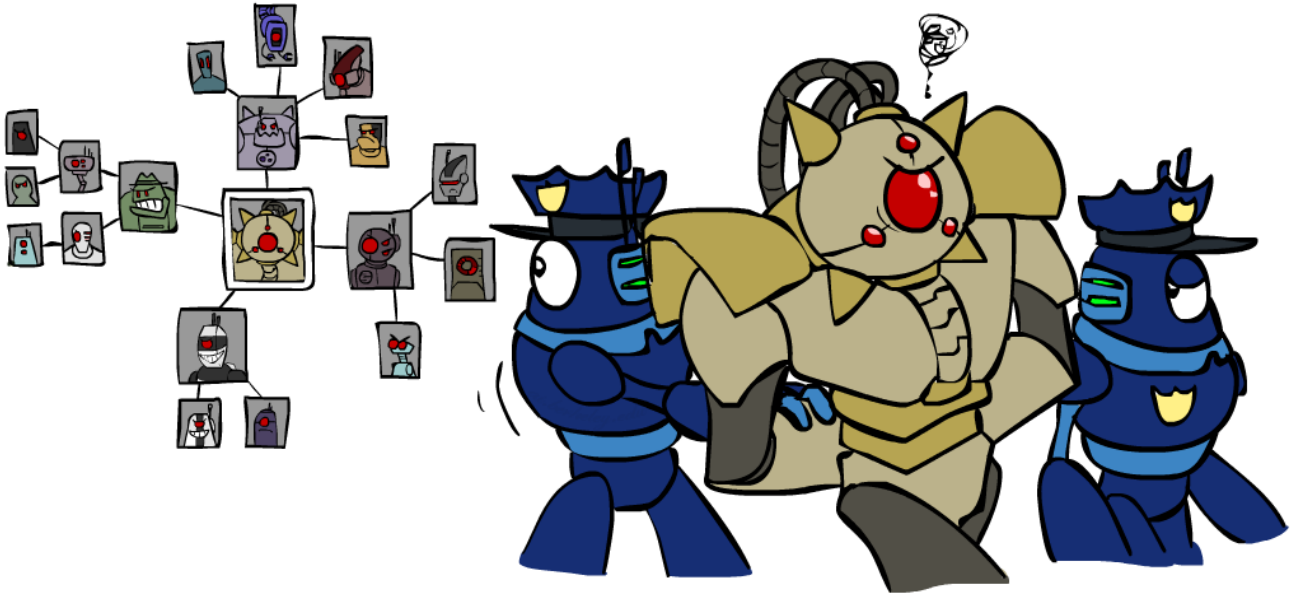
Tree-Structured CSPs

- 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)



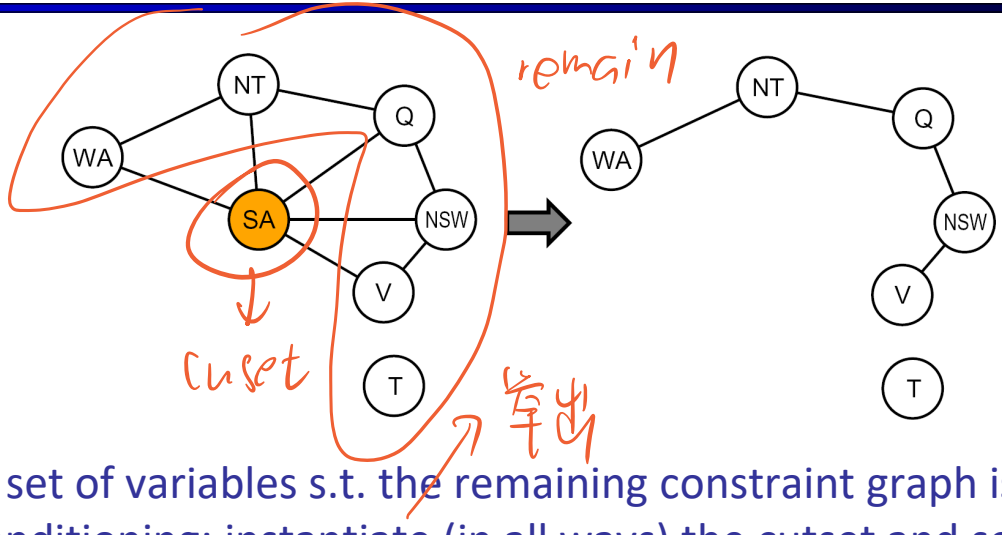
- Claim 2: If root-to-leaf arcs are consistent, forward assignment will not backtrack
- Easy to prove
- Why doesn't this algorithm work with cycles in the constraint graph?
- Note: we'll see this basic idea again with Bayes' nets

Cutset Conditioning



To make it a tree

Nearly Tree-Structured CSPs



- Cutset: a set of variables s.t. the remaining constraint graph is a tree
- Cutset conditioning: instantiate (in all ways) the cutset and solve the remaining tree-structured CSP
  - Cutset size c gives runtime $O((d^c) (n-c) d^2)$, very fast for small c

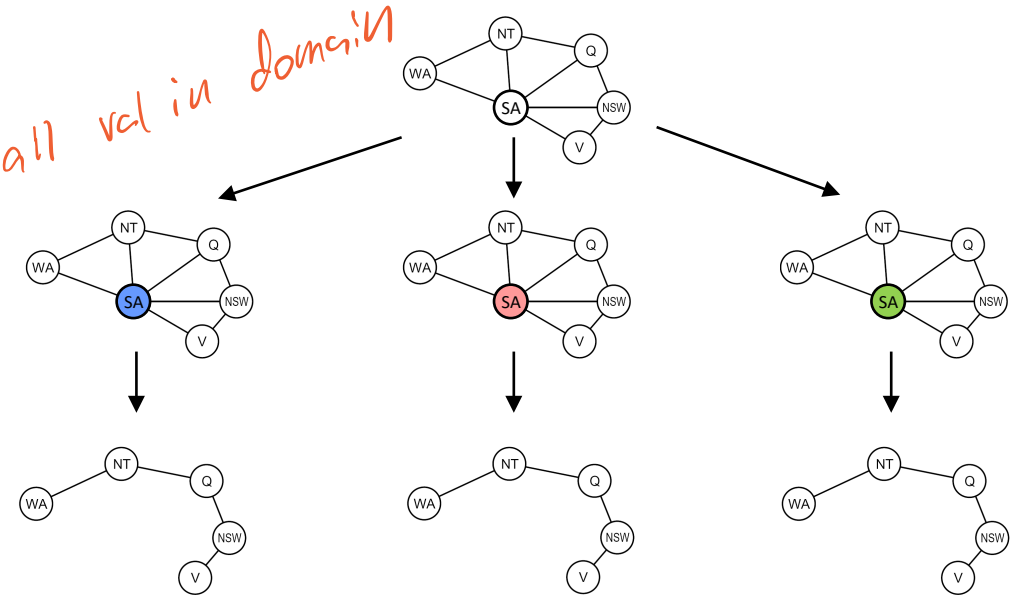
Cutset Conditioning

Choose a cutset

Instantiate the cutset
(all possible ways)

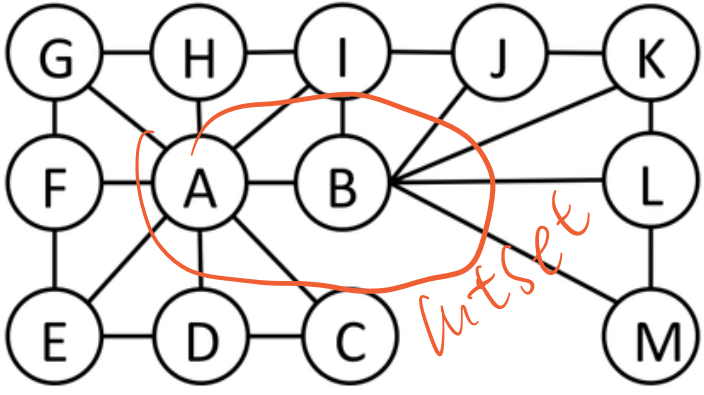
Compute residual CSP
for each assignment

Solve the residual CSPs
(tree structured)



Finding Cutset

- Find the smallest cutset for the graph below.



- Finding the *smallest* cutset is NP-hard
- But there are efficient approximation algorithms