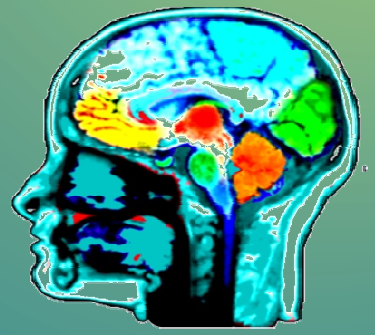
Introduction
To
Artificial Intelligence

Isfahan University of Technology (IUT)

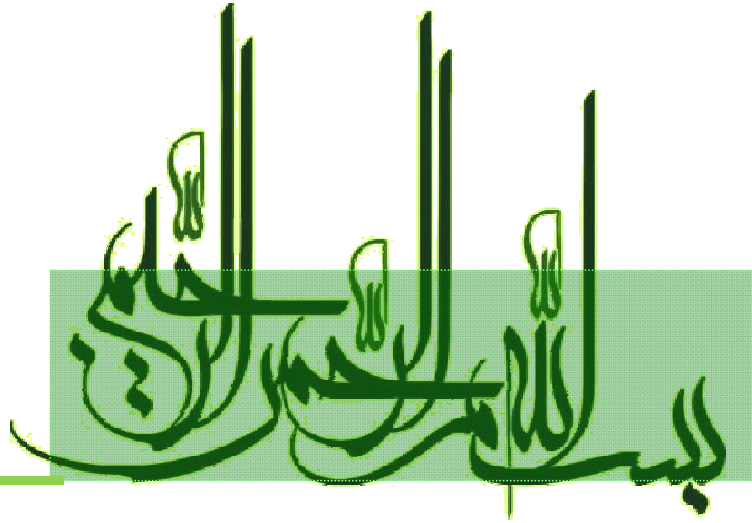
1402



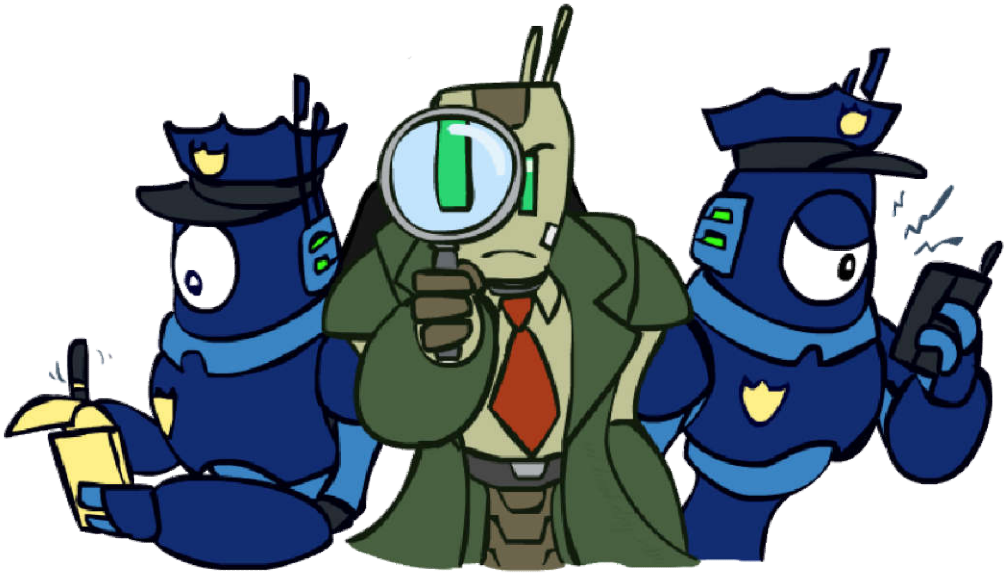
Constraint Satisfaction Problems

Dr. Hamidreza Hakim
hakim@iut.ac.ir

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley.]

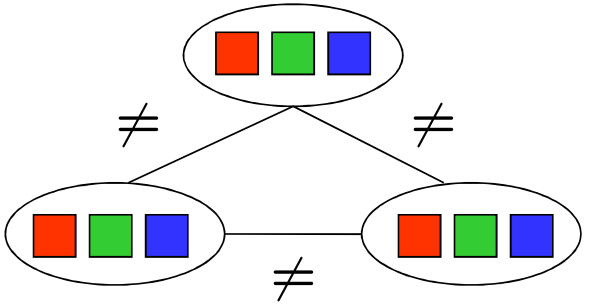

بسم الله الرحمن الرحيم

Constraint Satisfaction Problems

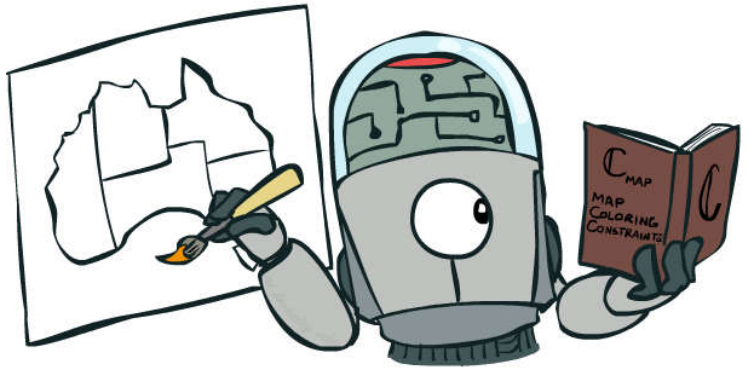


Reminder: CSPs

- 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.



Improving Backtracking

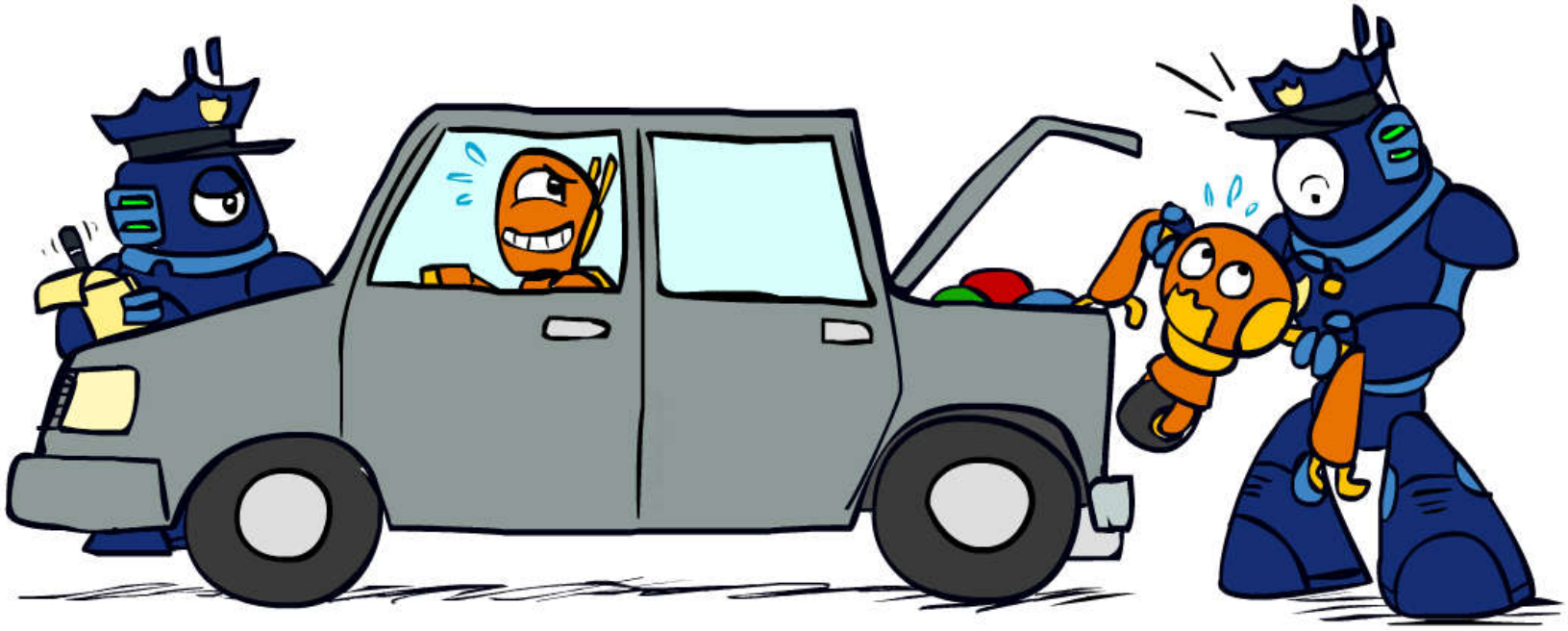
- General-purpose ideas give huge gains in speed
  - … but it's all still NP-hard

- Ordering:
  - Which variable should be assigned next? (MRV)
  - In what order should its values be tried? (LCV)
  - [In what order should constraints be checked (FFP)]

- Filtering: Can we detect inevitable failure early?

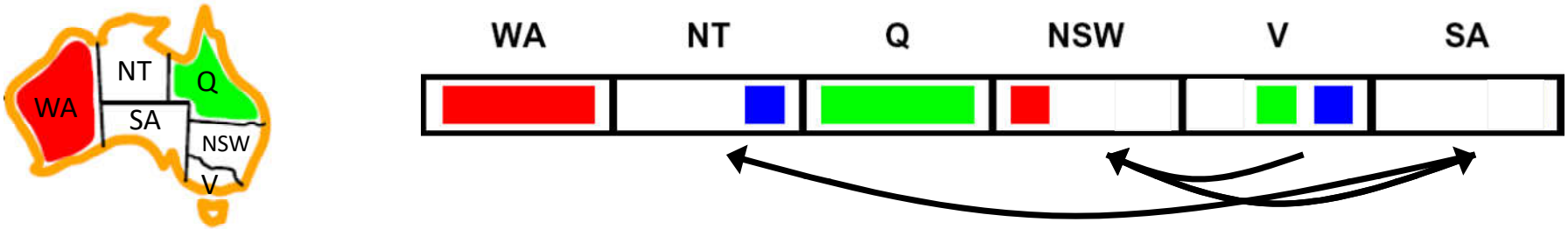
- Structure: Can we exploit the problem structure?

Arc Consistency and Beyond



Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are simultaneously consistent:

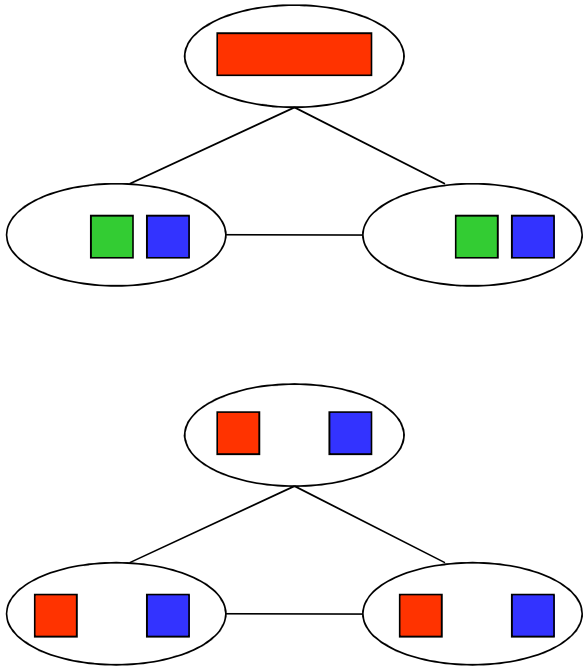


- Arc consistency detects failure earlier than forward checking
- Important: If X loses a value, neighbors of X need to be rechecked!
- Must rerun after each assignment!

Remember: Delete from the tail!

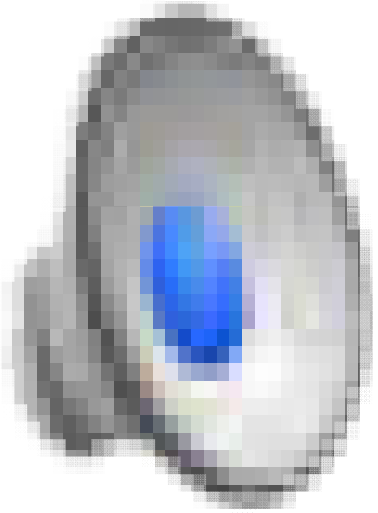
Limitations of Arc Consistency

- 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!

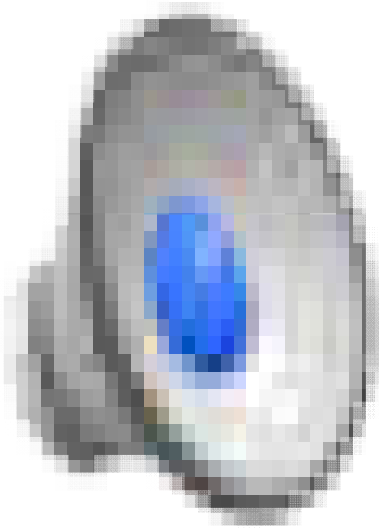


What went wrong here?

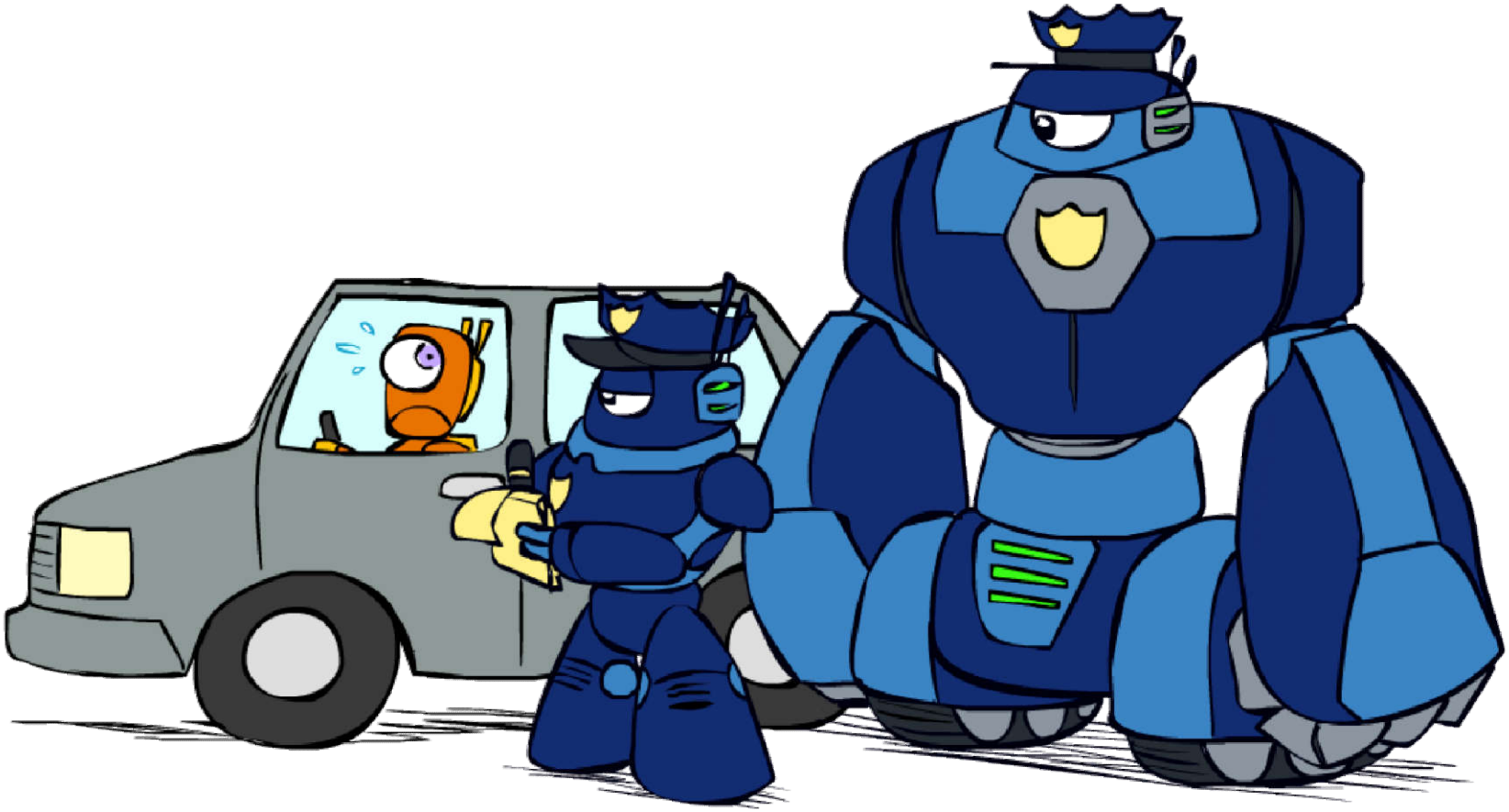
Video of Demo Coloring – Backtracking with Forward Checking – Complex Graph



Video of Demo Coloring – Backtracking with Arc Consistency – Complex Graph

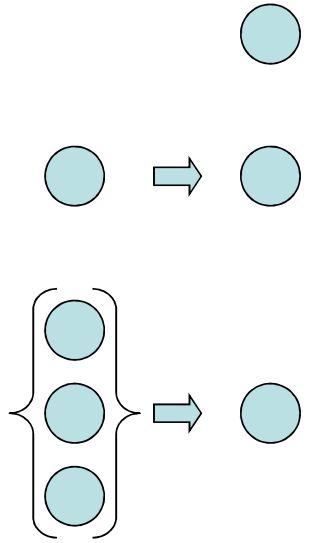


K-Consistency

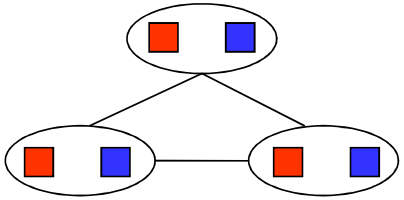


K-Consistency

- 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.(path consistency)



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



Strong K-Consistency

- Strong k-consistency:

Graph is 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

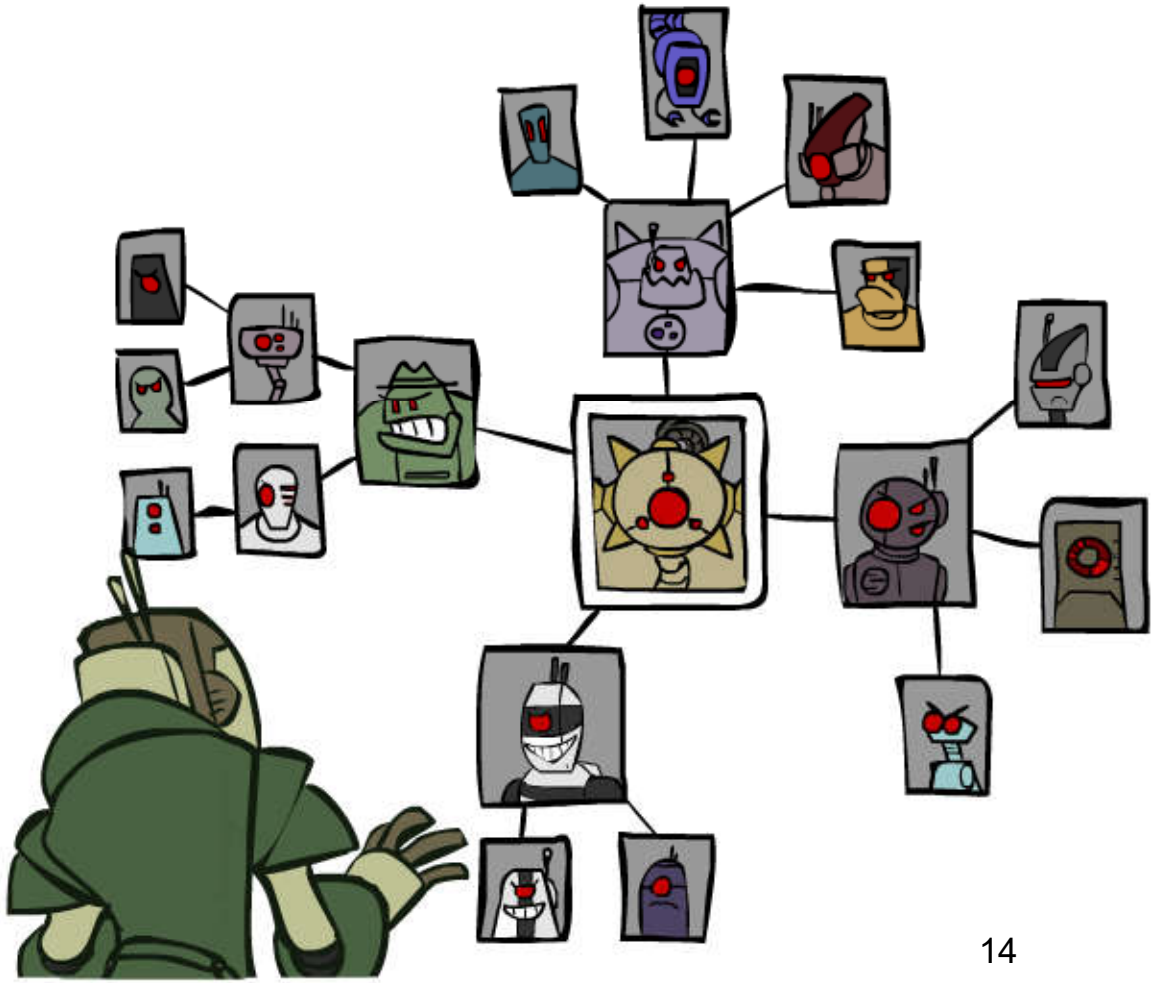
  - Choose a new variable

  By 3-consistency, there is a choice consistent with the first 2

  - …

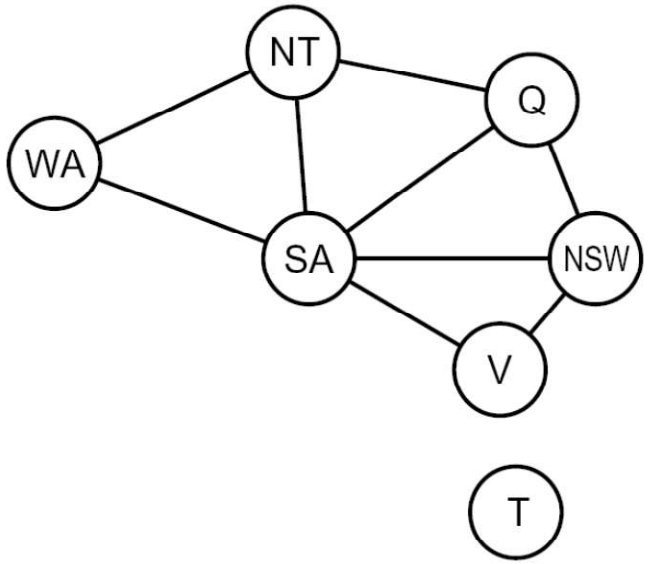
- Lots of middle ground between arc consistency and n-consistency (e.g. k=3, called path consistency)

Structure

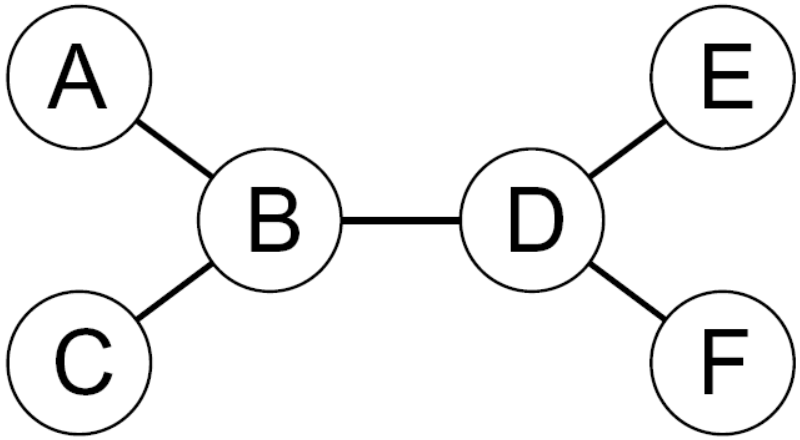


Problem Structure

- 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



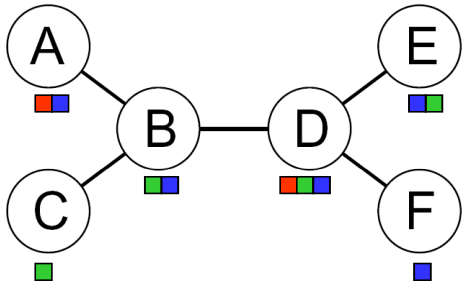
Tree-Structured CSPs

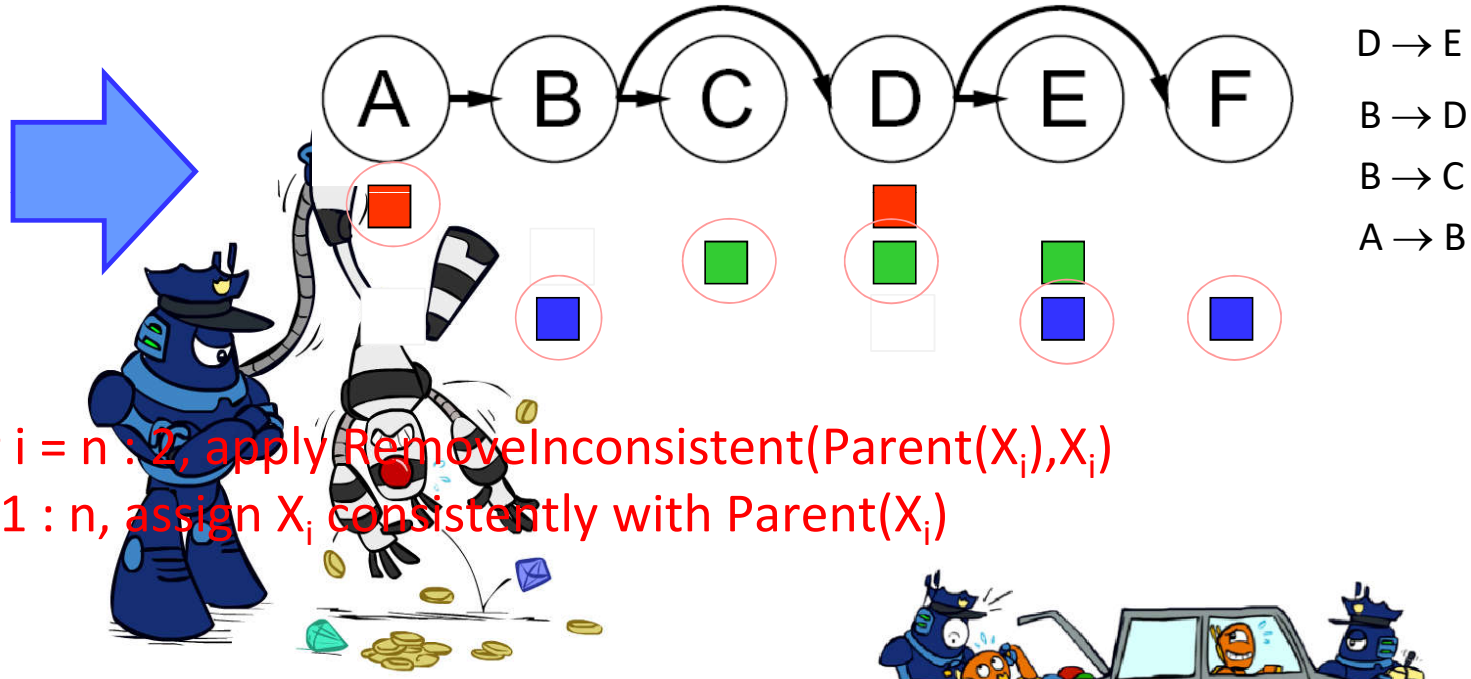


- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\ d^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$
- This property also applies to probabilistic reasoning (later): an example of the relation between syntactic restrictions and the complexity of reasoning

Tree-Structured CSPs

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



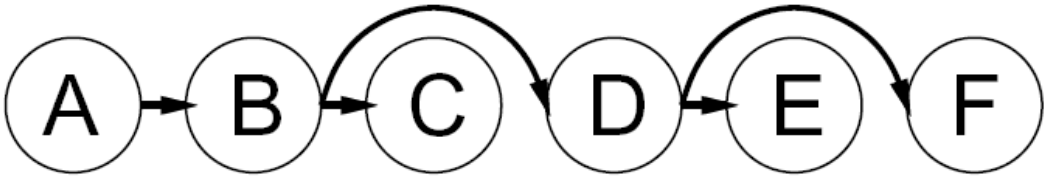


  - Remove backward: For i = n : 2, apply RemoveInconsistent(Parent(X_i),X_i)
  - Assign forward: For i = 1 : n, assign X_i consistently with Parent(X_i)

- Runtime: O(n d^2)

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: this basic idea again with Bayes' nets

Tree-Structured CSPs Algorithm

function $TREE\_CSP\_SOLVER(csp)$ **returns** a solution or failure

input: csp, a CSP with components X,D, C

$n \leftarrow$ number of variables in X
$assignment \leftarrow$ an empty assignment
$root \leftarrow$ any variable in X
$X \leftarrow TOPOLOGICAL(X, root)$
for $j = n$ **down to** 2 **do**
    $MAKE\_ARC\_CONSISTENT(PARENT(X_j), X_j))$
    if it cannot be made consistent **then return** $failure$
for $i = 1$ **to** n **do**
    $assignment[X_i] \leftarrow anyconsistent\ value\ from\ D_i$
    if there is no consistent value **then return** $failure$
return $assignment$

Tree-Structured CSPs Complexity

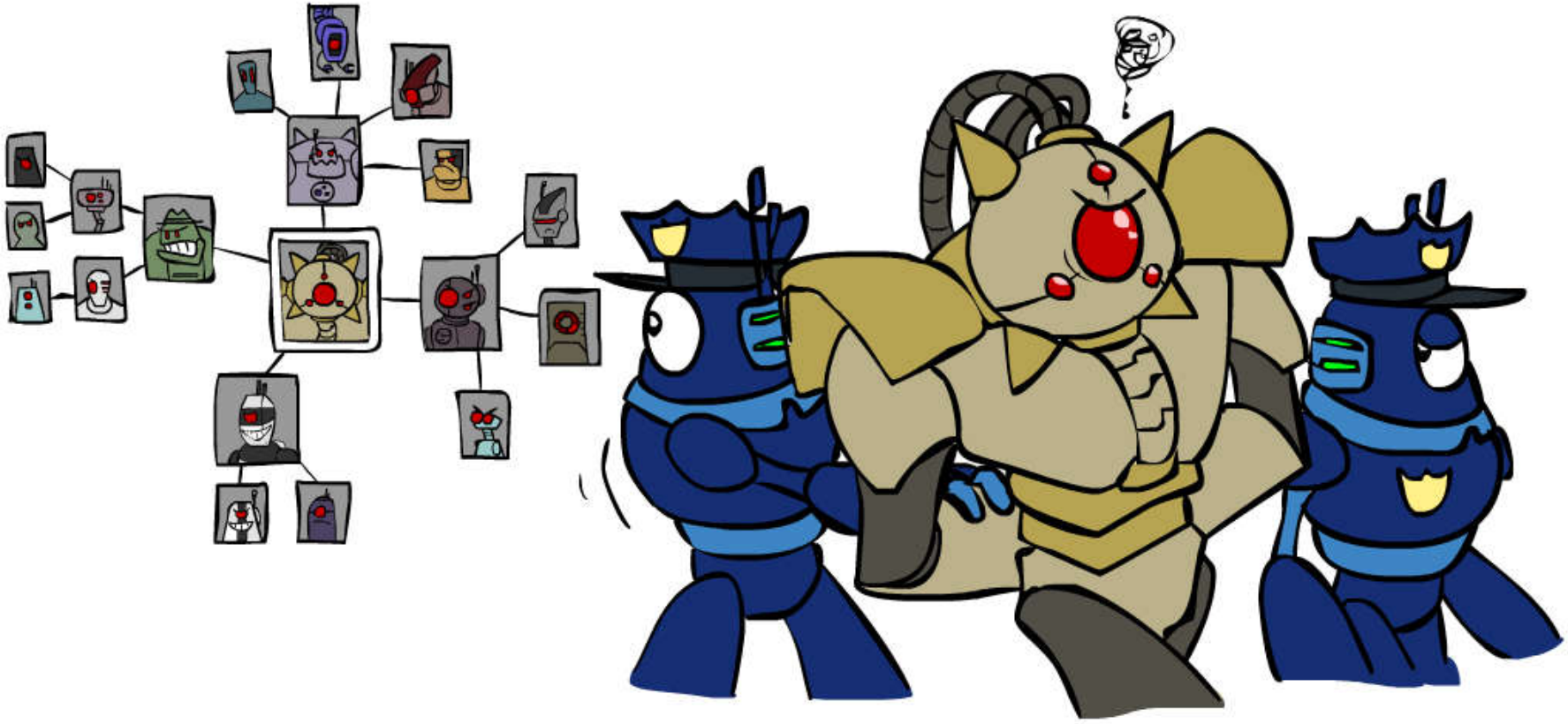
- Linear Complexity

$X \leftarrow$ Topological Sort
for $i = n$ downto 2 do
    Make-Arc-Consistent(Parent(X_i), X_i)
for $i = 1$ to n do
    $X_i \leftarrow$ any remaining value in D_i

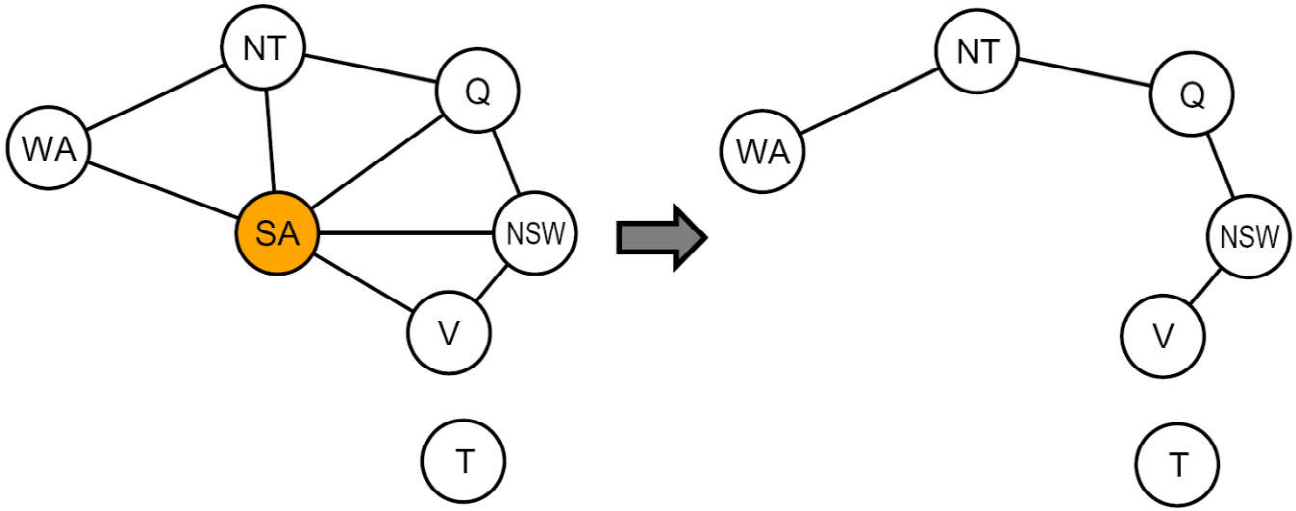
⇒ remove all values from domain of Parent(X_i) which may violate arc-consistency

- After running loop1, any arc from a parent to its child is arc-consistent.
- ⇒ if the constraint graph has no loops, the CSP can be solved in $O(nd^2)$ time.

Improving Structure



Nearly Tree-Structured CSPs



How to create Tree?

- 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
- Complexity Order?

1) Find a subset S such that the remaining graph becomes a tree
2) For each possible consistent assignment to S
   a) remove inconsistent values from domains of remaining variables
   b) solve the remaining CSP which has a tree structure

- Cutset size c gives runtime O((d^c) (n-c) d^2), very fast for small c
- Why?
- D value for cutset -> d c
- (N-C)d2

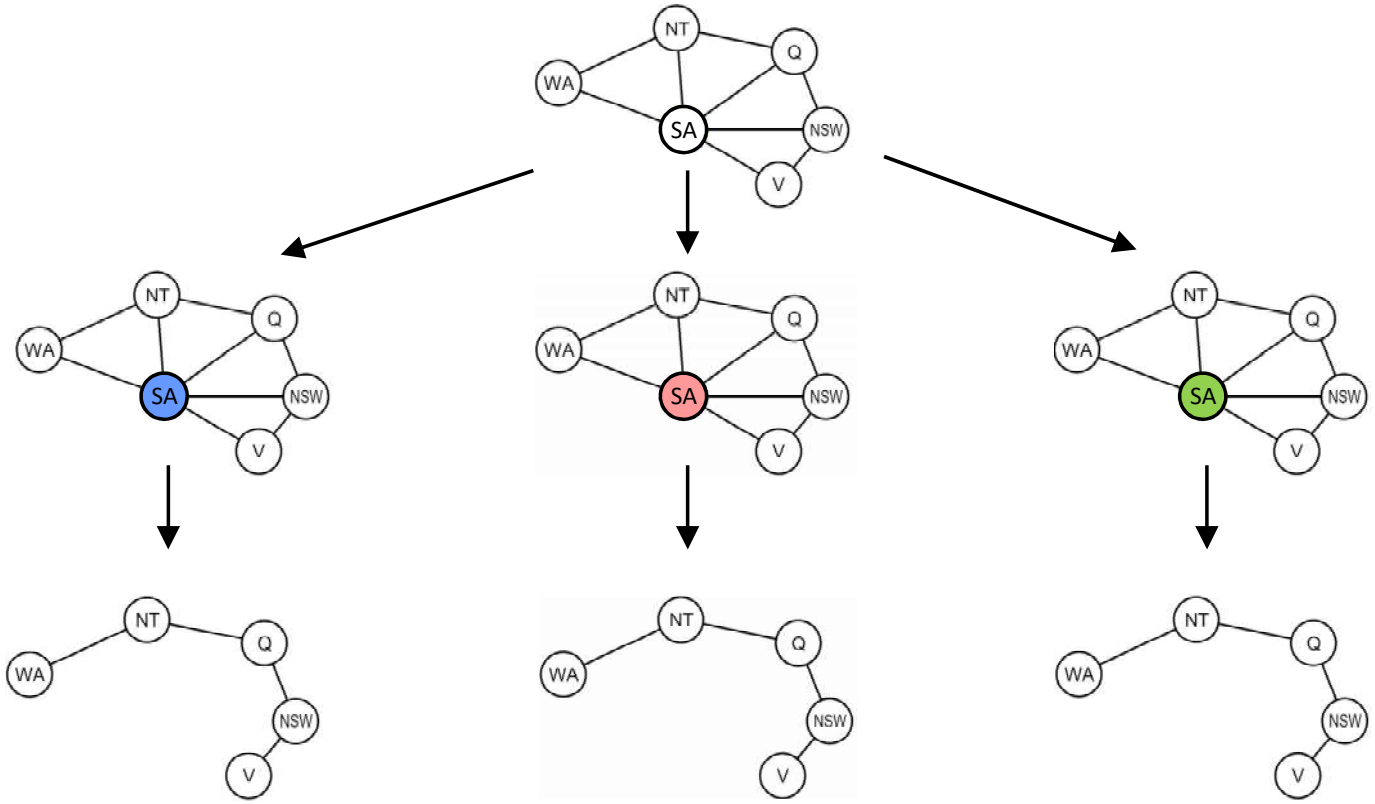
Cutset Conditioning

Choose a cutset

Instantiate the cutset (all possible ways)

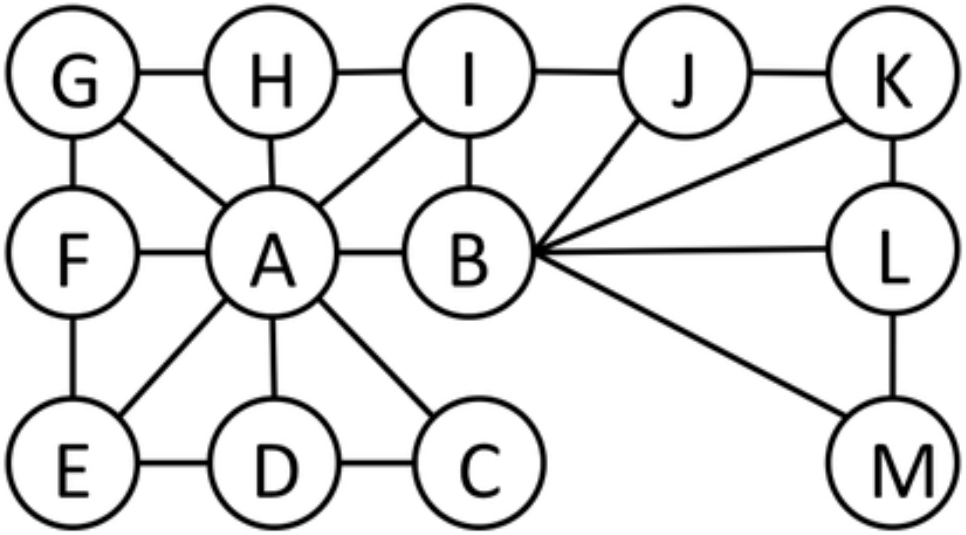
Compute residual CSP for each assignment

Solve the residual CSPs (tree structured)



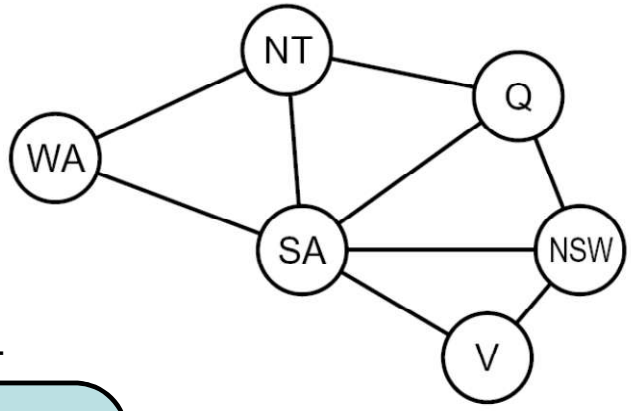
Cutset Quiz

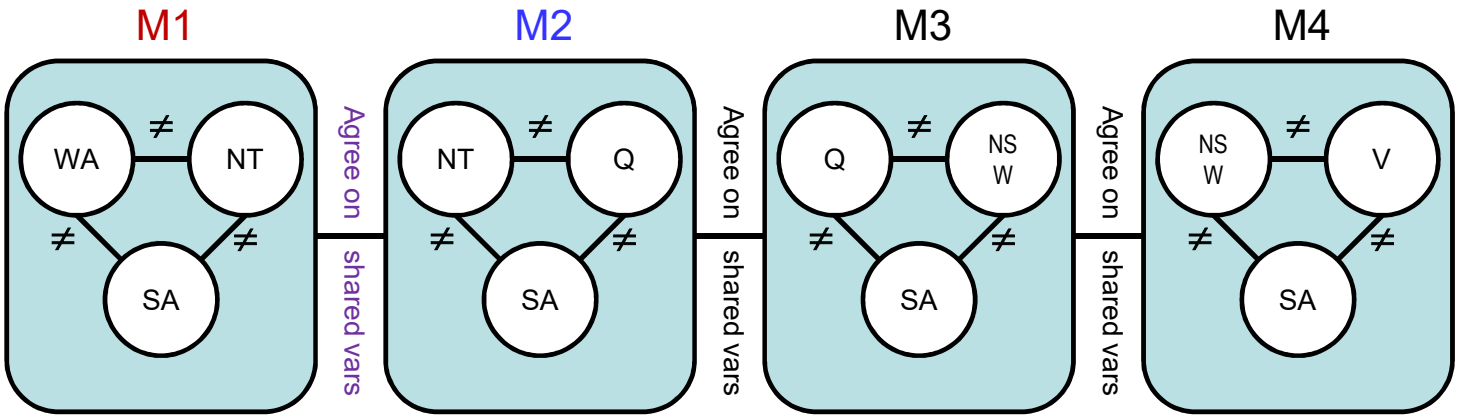
- Find the smallest cutset for the graph below.



Tree Decomposition*

- Idea: create a tree-structured graph of mega-variables
- Each mega-variable encodes part of the original CSP
- Sub problems overlap to ensure consistent solutions





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

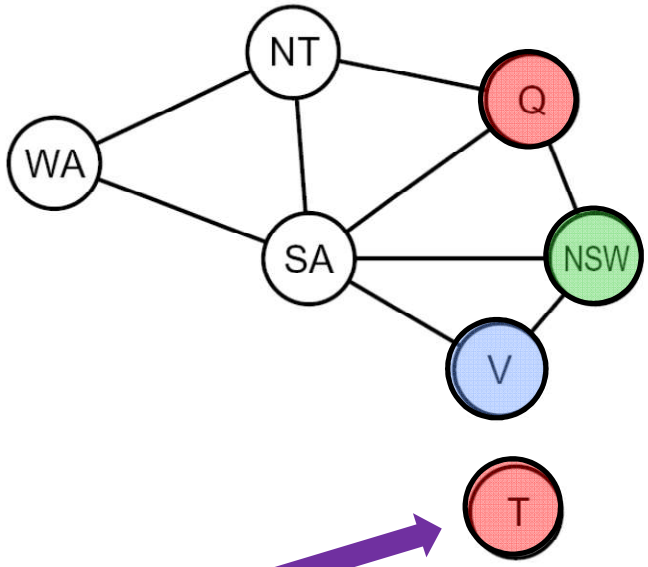
{(NT=r,SA=g,Q=b), (NT=b,SA=g,Q=r), …}

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

Intelligent Backtracking

- Q ← Red
- NSW ← Green
- V ← Blue
- T ← Red
- SA ← ?

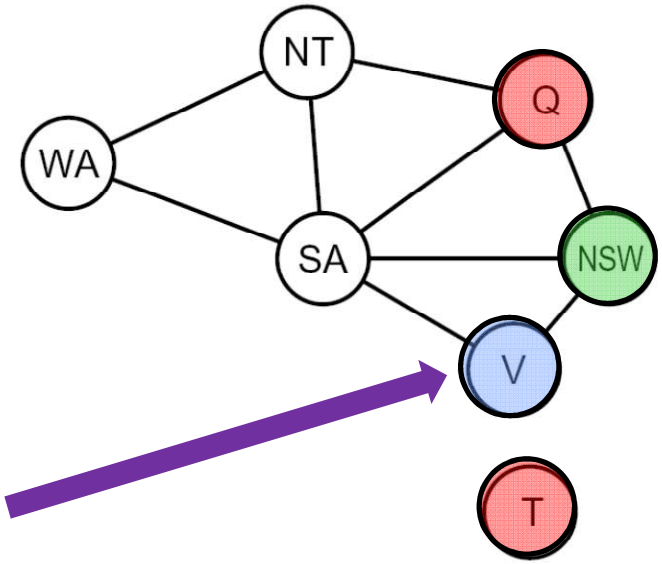
- Chronological Backtracking



Backtracks to

Back Jumping

- Q ← Red
- NSW ← Green
- V ← Blue
- T ← Red
- SA ← ?

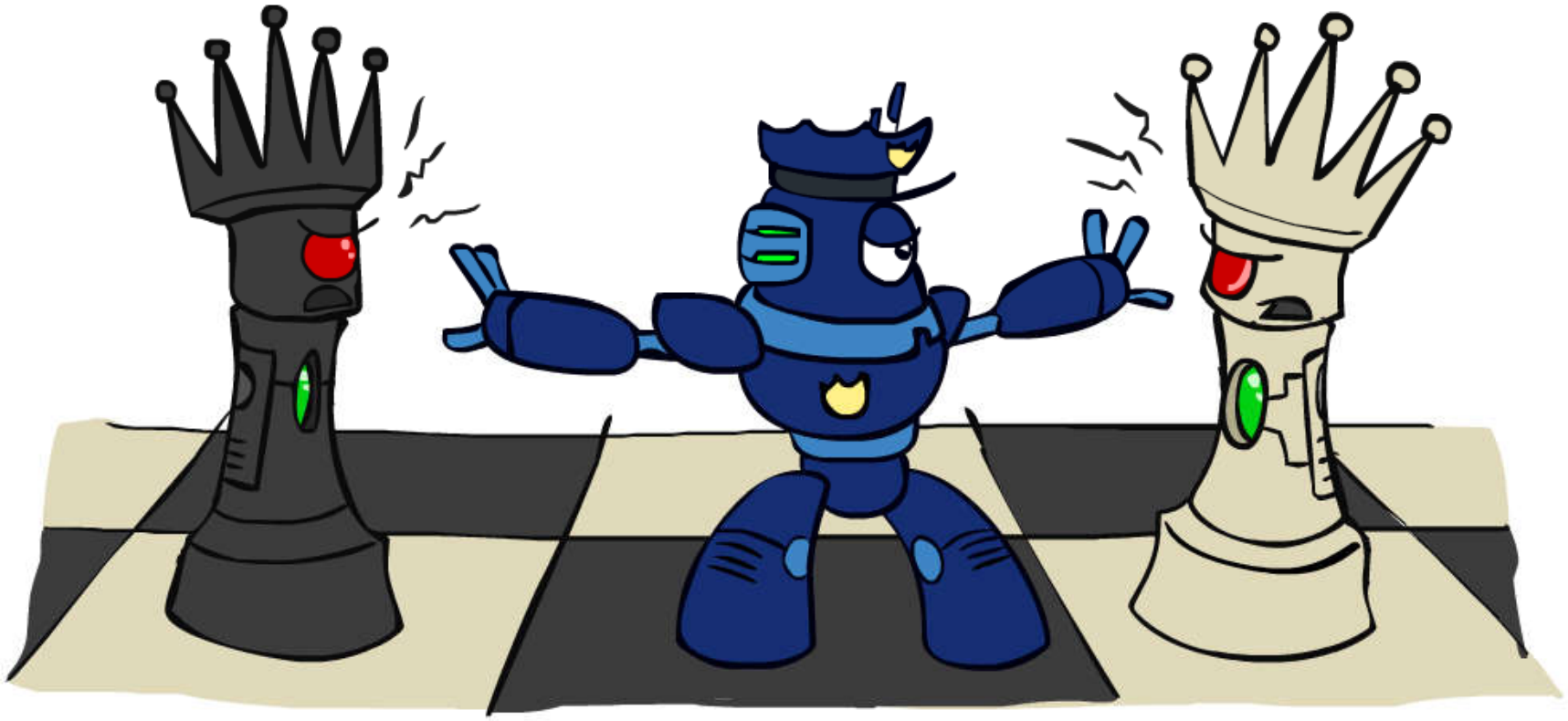


- Keeps a conflict set (i.e., previously colored neighbors) for each node

Backtracks to

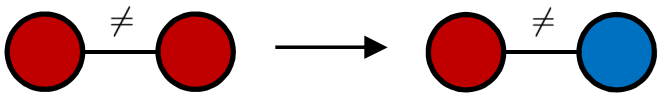
- E.g. conflict set of SA = {Q, NSW, V}

Iterative Improvement



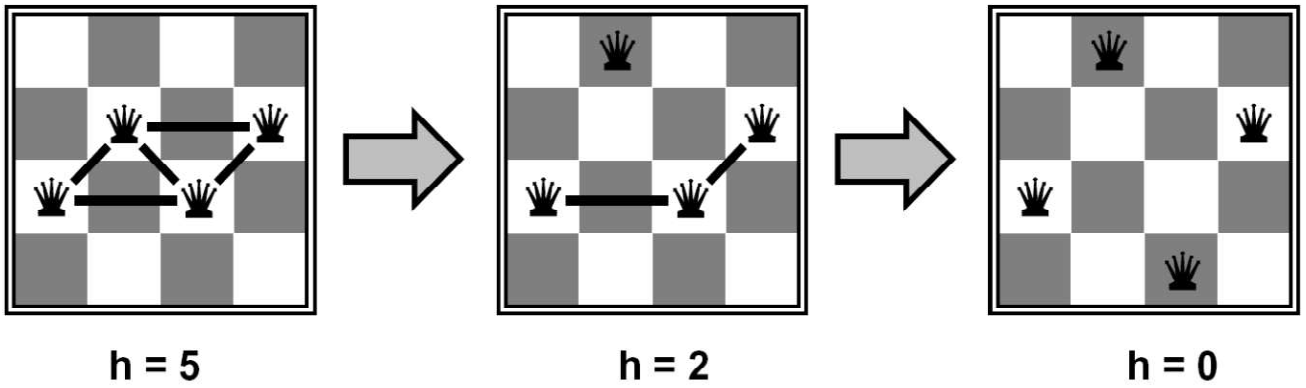
Iterative Algorithms for CSPs

- 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.



- Min-Conflicts 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
    - Greedy + Randomness like Local Seaech

Example: 4-Queens

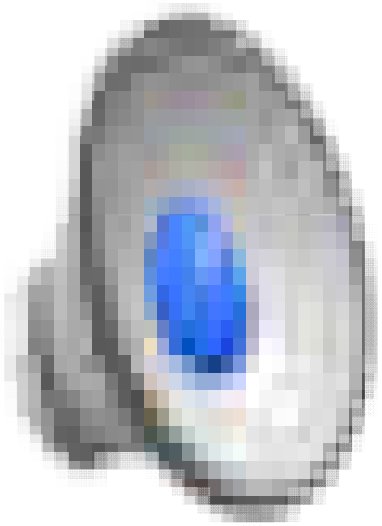


- States: 4 queens in 4 columns ($4^4 = 256$ states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: c(n) = number of attacks

[Demo: n-queens – iterative improvement (L5D1)]
[Demo: coloring – iterative improvement]

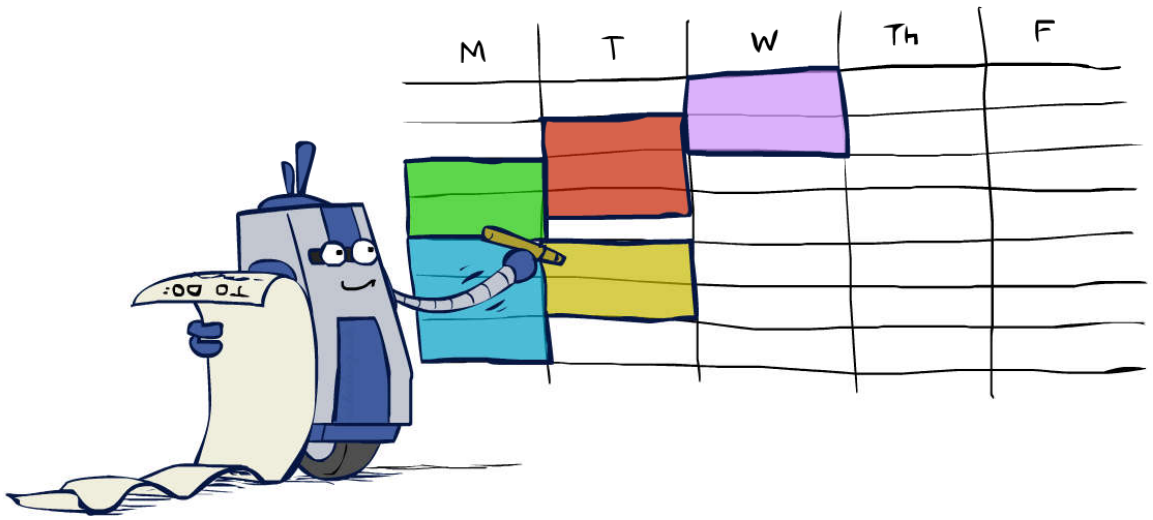
Video of Demo Iterative Improvement – n Queens

Video of Demo Iterative Improvement – Coloring



Summary: CSPs

- CSPs are a special kind of search problem:
  - States are partial assignments
  - Goal test defined by constrai
- Basic solution: backtracking sea
- Speed-ups:
  - Ordering
  - Filtering
  - Structure

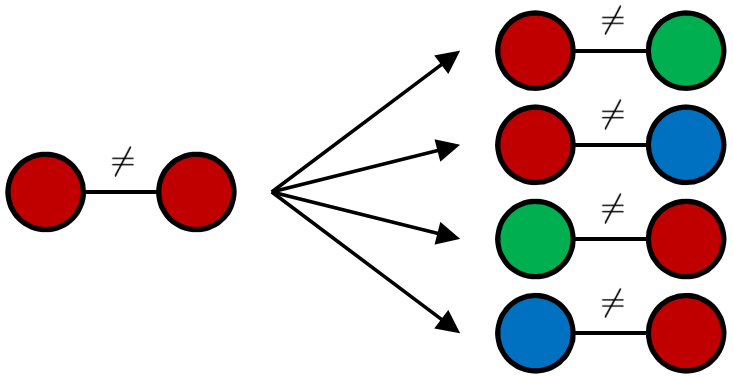


- Iterative min-conflicts is often effective in practice

Local Search

Local Search

- Tree search keeps unexplored alternatives on the fringe (ensures completeness)
- Local search: improve a single option until you can't make it better (no fringe!)
- New successor function: local changes



- Generally much faster and more memory efficient (but incomplete and suboptimal)

Next Time: Adversarial Search!