CSE 473: Artificial Intelligence

Autumn 2018

Constraint Satisfaction Problems - Part 2

Steve Tanimoto

des from :
ox, Dan Weld, Dan Klein, Stuart Russell, Andrew Moore, Luke Zettlemoyer

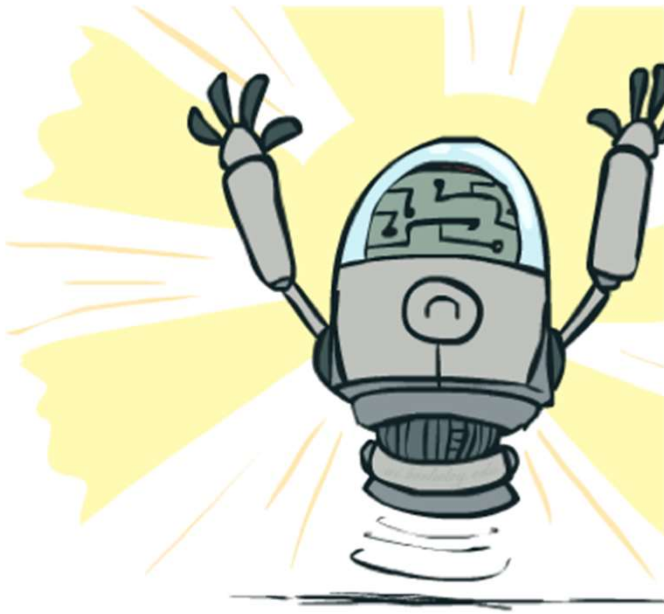
Improving Backtracking

- eneral-purpose ideas give huge gains in speed

- rdering:
  - Which variable should be assigned next?
  - In what order should its values be tried?

- ltering: Can we detect inevitable failure early?

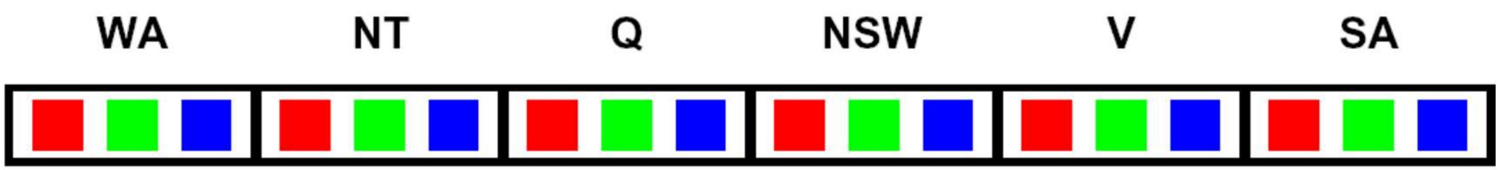
- tructure: Can we exploit the problem structure?



Filtering

Filtering: Forward Checking

- ltering: Keep track of domains for unassigned variables and cross off bad options
- orward checking: Cross off values that violate a constraint when added to the exis
ssignment

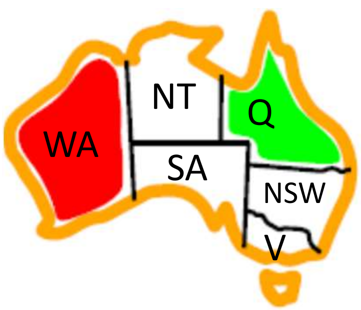


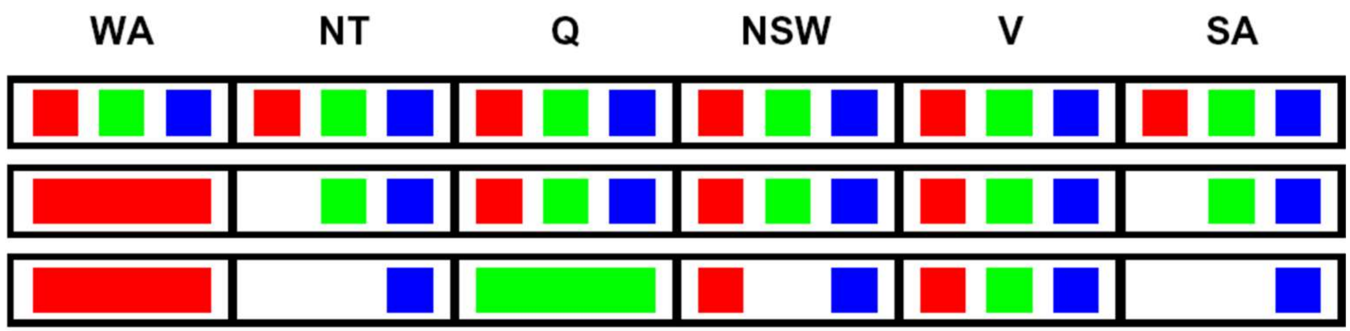
[Demo: coloring -- forwar

eo of Demo Coloring – Backtracking with Forward Chec

Filtering: Constraint Propagation

orward checking only propagates information from assigned to unassigned
doesn't catch when two unassigned variables have no consistent assignment:

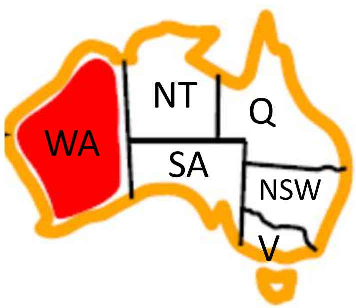


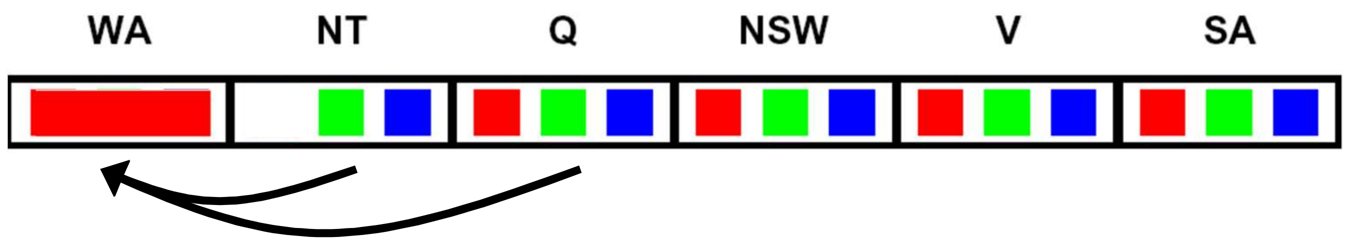


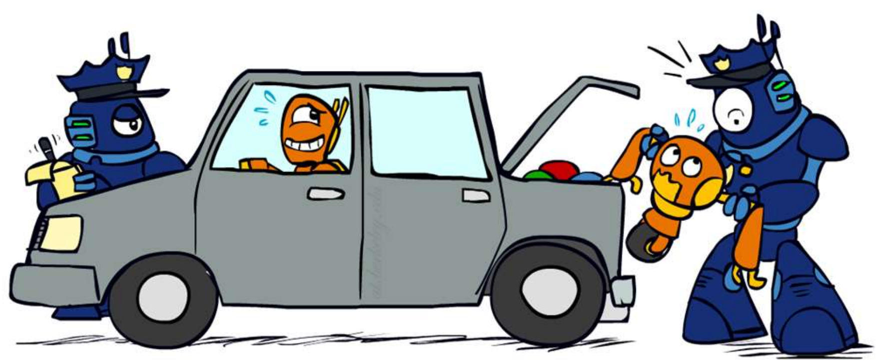
T and SA cannot both be blue!
hy didn't we detect this yet?
onstraint propagation: reason from constraint to constraint

Consistency of a Single Arc

n arc X $\rightarrow$ Y is consistent iff for *every* x in the tail there is *some* y in the head which ould be assigned without violating a constraint





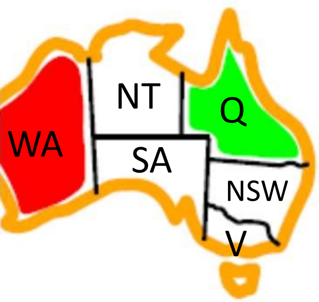


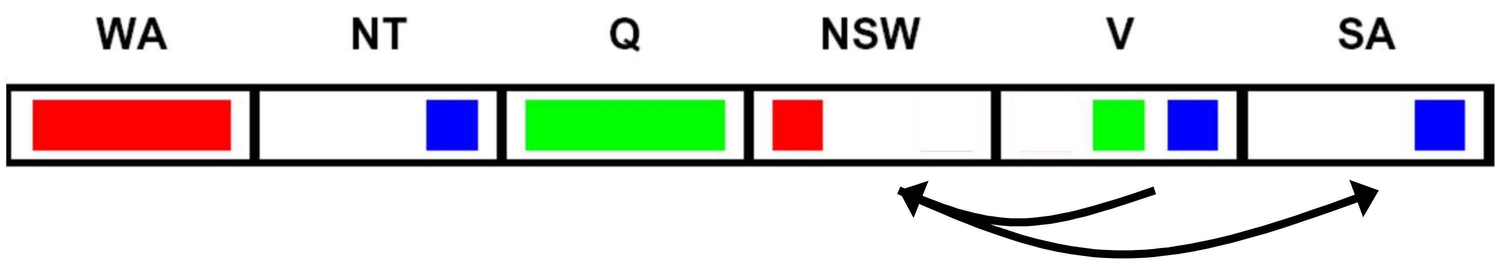
Delete from the tail!

orward checking: Enforcing consistency ***of arcs pointing to each new assignment***

Arc Consistency of an **Entire CSP**

simple form of propagation makes sure all arcs are consistent:





- nportant: If X loses a value, neighbors of X need to be rechecked!
- rc consistency detects failure ***earlier*** than forward checking
- an be run as a preprocessor ***or*** after each assignment
- /hat's the ***downside*** of enforcing arc consistency?

Remember: Delete
from the tail!

AC-3 algorithm for Arc Consistency

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

  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*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 ← *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* ← *true*
  return *removed*

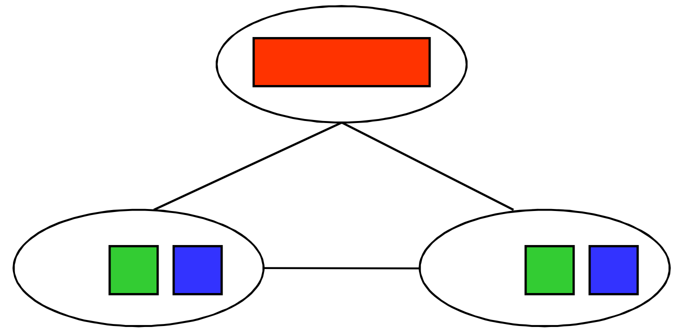
- Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$
- … but detecting ***all*** possible future problems is NP-hard – why?

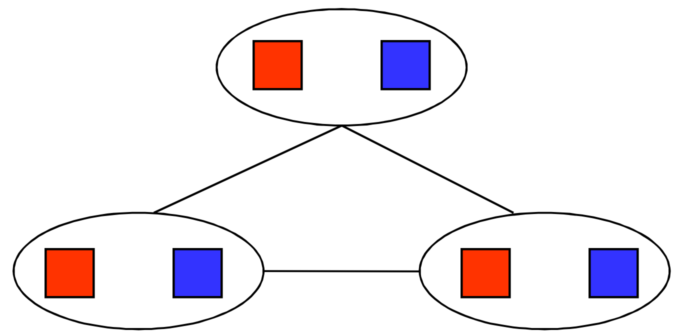
[Demo: CSP applet (made available by aispace.org) -- r

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!



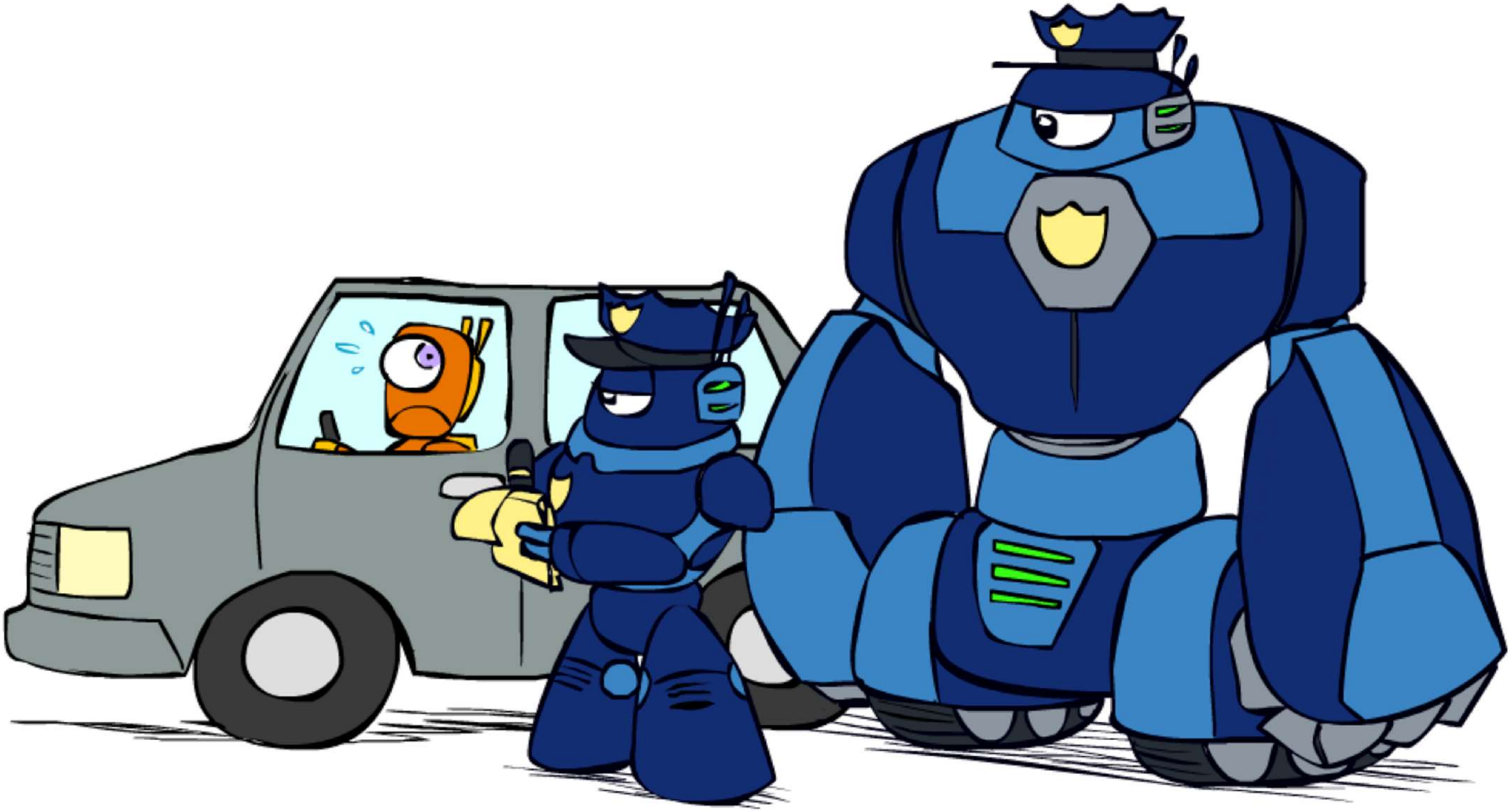


What went wrong here?

[Demo: coloring -- forward cl

[Demo: coloring -- arc consis

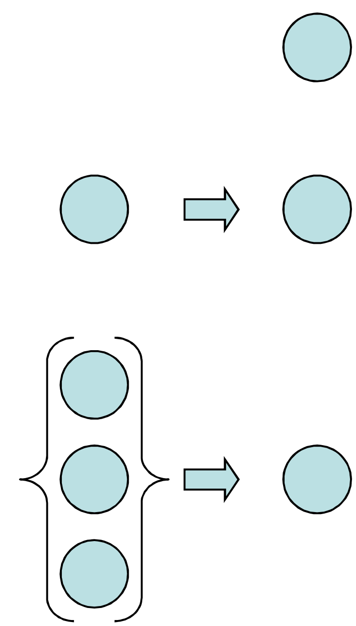
K-Consistency



K-Consistency

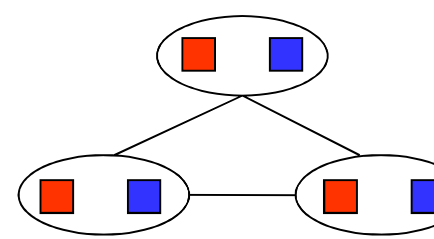
ncreasing degrees of consistency

- 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.



igher k more expensive to compute

ou need to know the algorithm for k=2 case: arc consistency)



Strong K-Consistency

- rong k-consistency: also k-1, k-2, … 1 consistent
- aim: strong n-consistency means we can solve without backtracking!
- /hy?
  - 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
  - …
- ots of middle ground between arc consistency and n-consistency! (e.g. k=3, called
ath consistency)

eo of Demo Arc Consistency – CSP Applet – n Que

o of Demo Coloring – Backtracking with Forward Check
Complex Graph

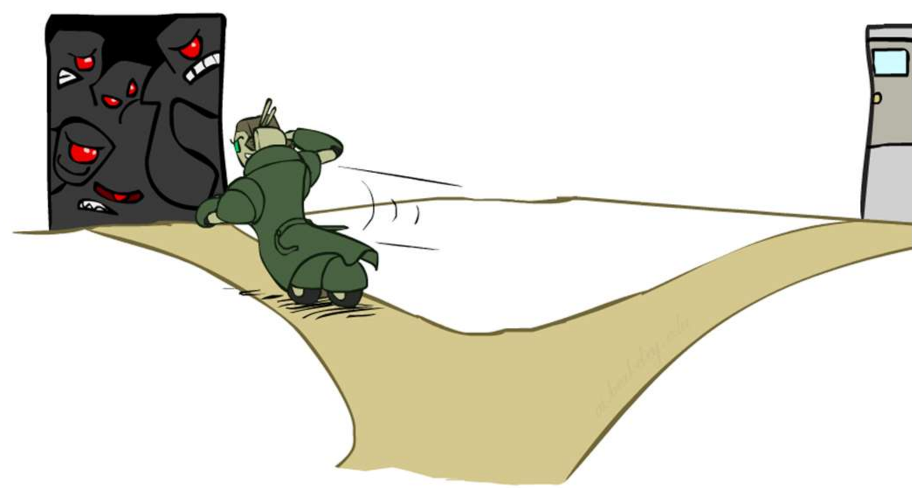
eo of Demo Coloring – Backtracking with Arc Consisten
Complex Graph

Ordering

Ordering: Minimum Remaining Values

ariable Ordering: Minimum remaining values (MRV):

- Choose the variable with the fewest legal left values in its domain



/hy min rather than max?

lso called “most constrained variable”

Fail-fast” ordering

Ordering: Maximum Degree

ie-breaker among MRV variables

- What is the very first state to color? (All have 3 values remaining.)

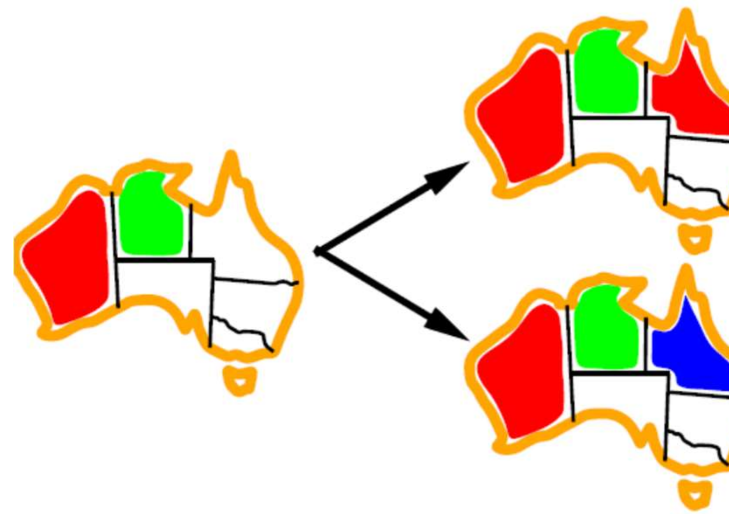
Maximum degree heuristic:

- Choose the variable participating in the most constraints on remainin variables

Why most rather than fewest constraints?

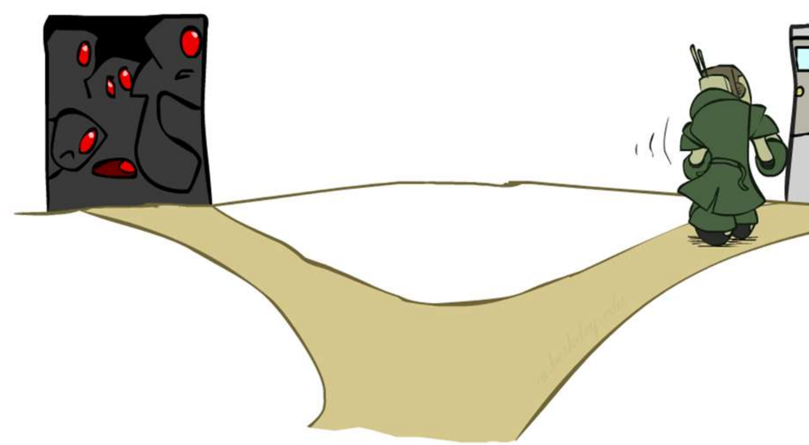
Ordering: Least Constraining Value

- /alue 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)



- Vhy least rather than most?

- ombining these ordering ideas makes 000 queens feasible



Rationale for MRV, MD, LCV

/e want to enter the most promising branch, but we also war
o detect failure quickly

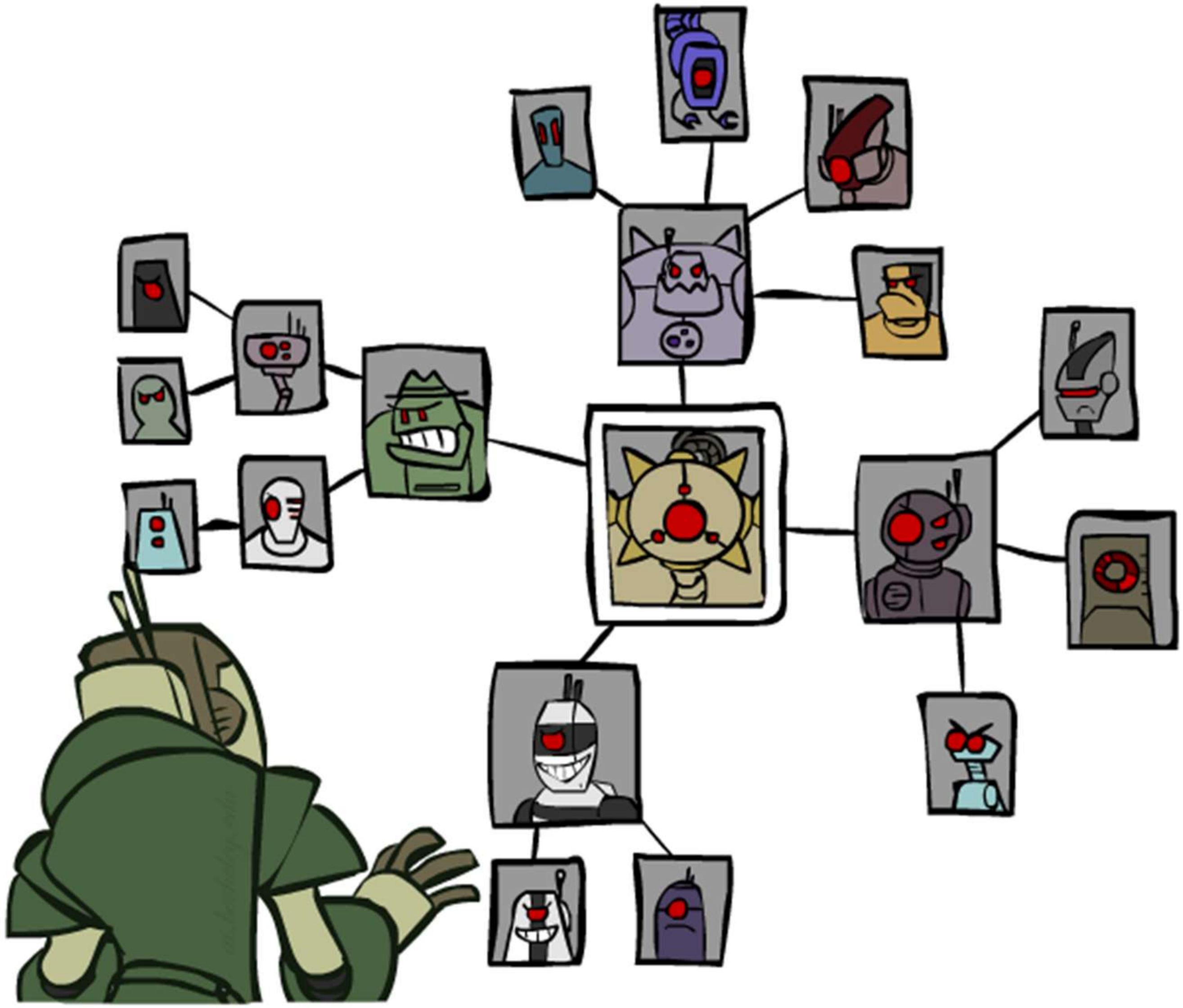
IRV+MD:

- Choose the variable that is most likely to cause failure
- It must be assigned at some point, so if it is doomed to fail, better to find out soon

CV:

- We hope our early value choices do not doom us to failure
- Choose the value that is most likely to succeed

Structure



Problem Structure

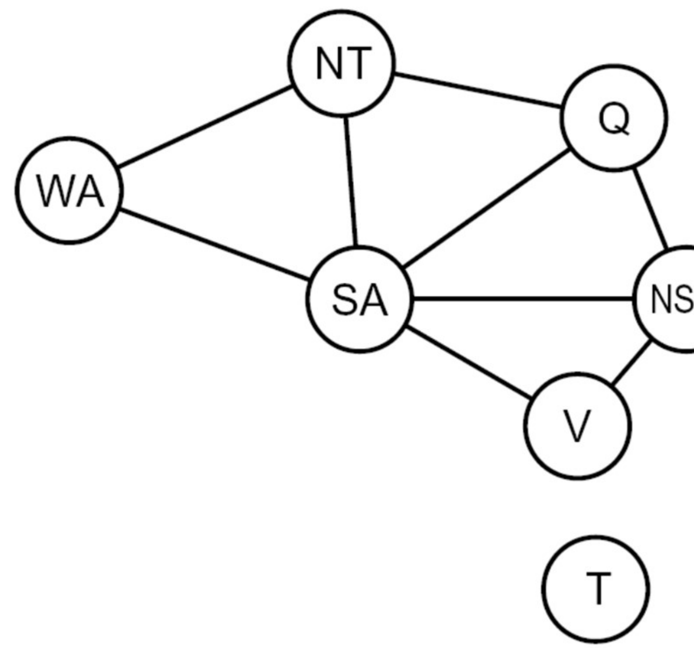
xtreme case: independent subproblems

- Example: Tasmania and mainland do not interact

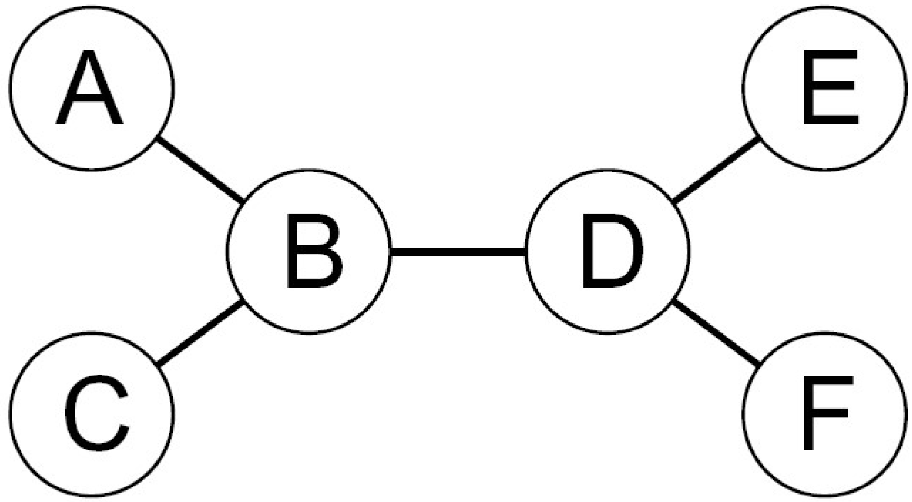
ndependent subproblems are identifiable as onnected components of constraint graph

uppose a graph of n variables can be broken into ubproblems 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



heorem: if the constraint graph has no loops, the CSP can be solved in $O(n\ d^2)$ time

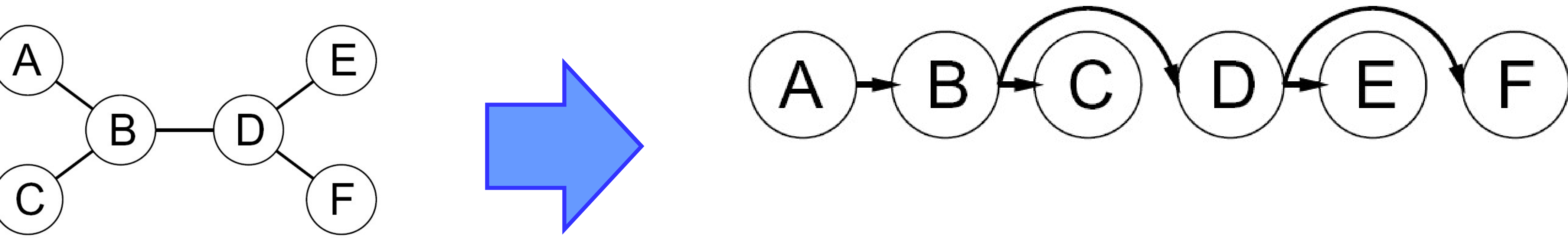
Compare to general CSPs, where worst-case time is $O(d^n)$

his property also applies to probabilistic reasoning (later): an example of the relati
etween syntactic restrictions and the complexity of reasoning

Tree-Structured CSPs

lgorithm for tree-structured CSPs:

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



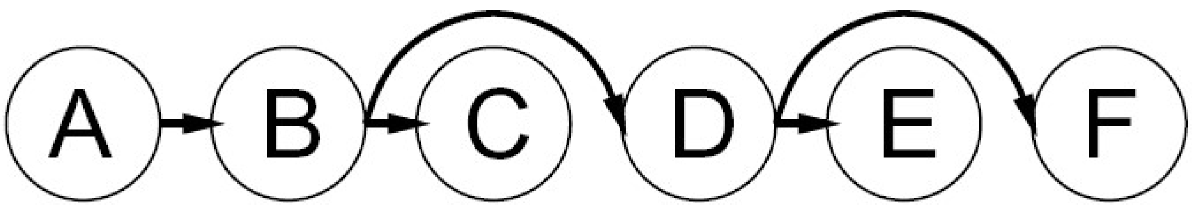
- Remove backward: For i = n : 2, apply RemoveInconsistent(Parent(X_i),X_i)
- Assign forward: For i = 1 : n, assign X_i consistently with Parent(X_i)

untime: O(n d^2) (why?)

Tree-Structured CSPs

laim 1: After backward pass, all root-to-leaf arcs are consistent

roof: Each X→Y was made consistent at one point and Y's domain could not have een reduced thereafter (because Y's children were processed before Y)



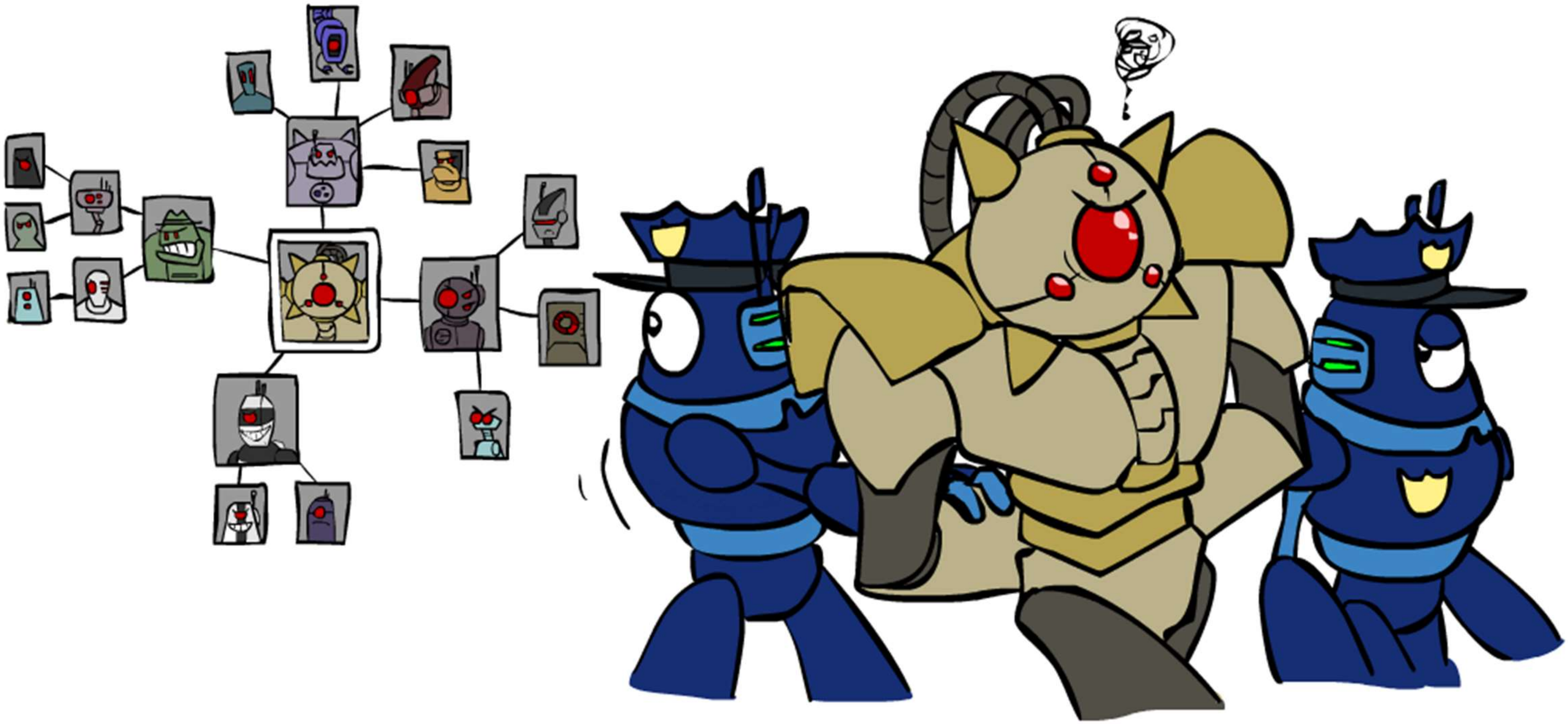
laim 2: If root-to-leaf arcs are consistent, forward assignment will not backtrack

roof: Induction on position

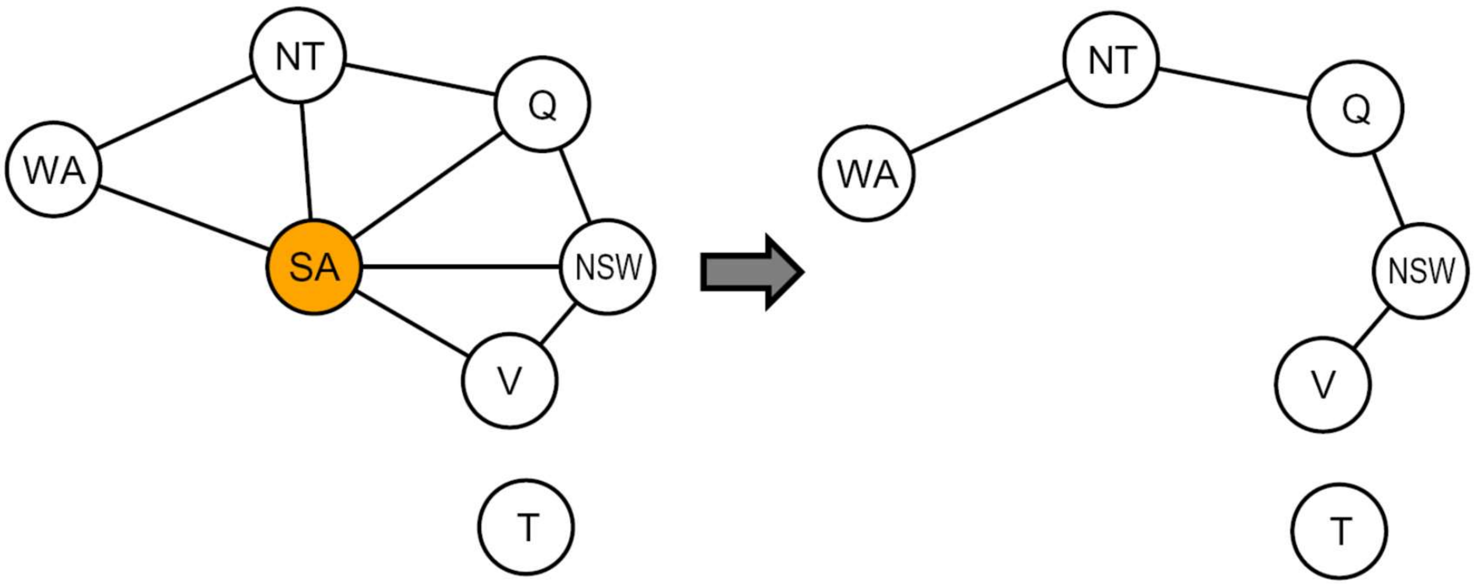
Vhy doesn't this algorithm work with cycles in the constraint graph?

ote: 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 $O((d^c) (n\text{-}c) d^2)$, very fast for small c

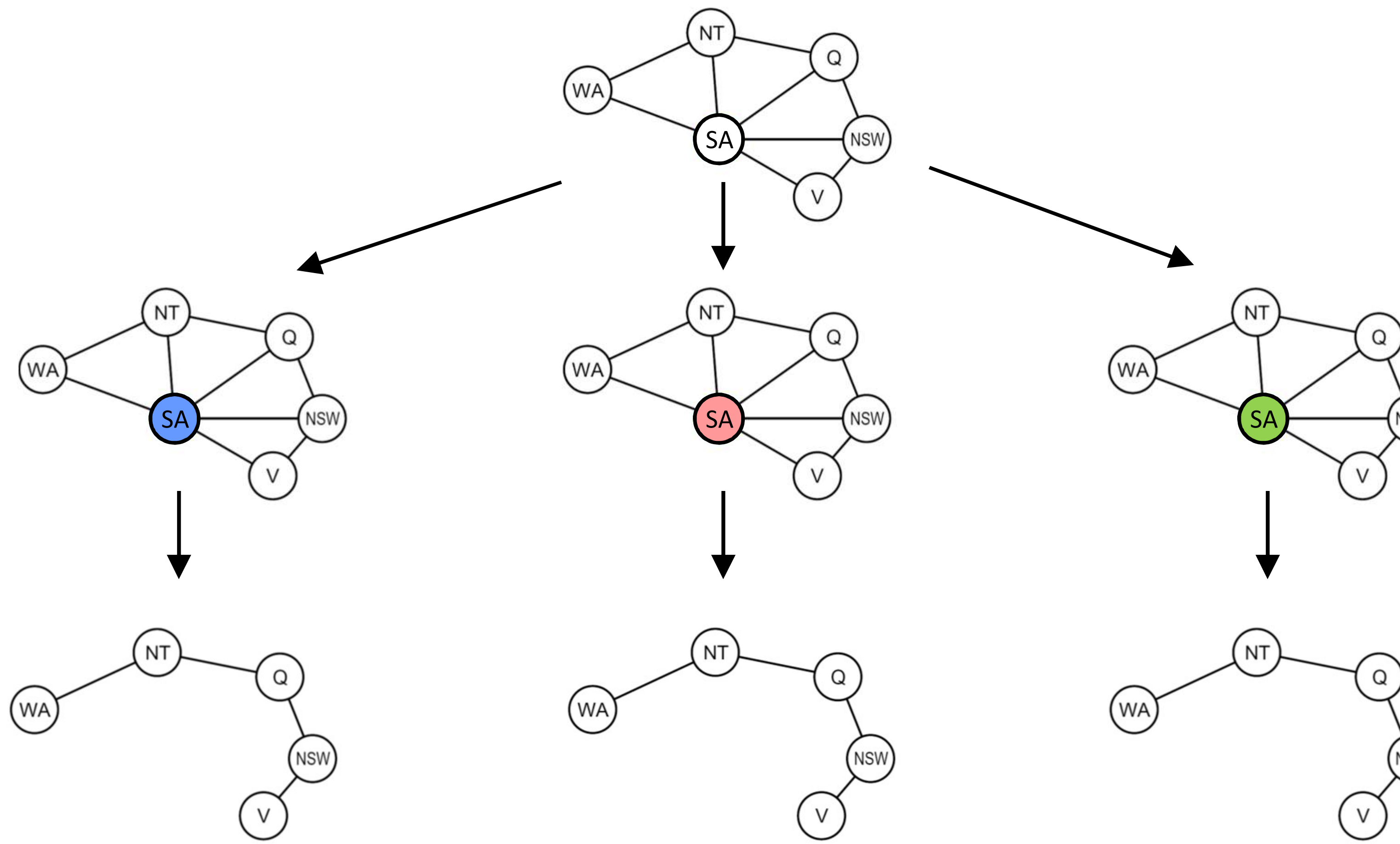
Cutset Conditioning

Choose a cutset

nstantiate the cutset
(all possible ways)

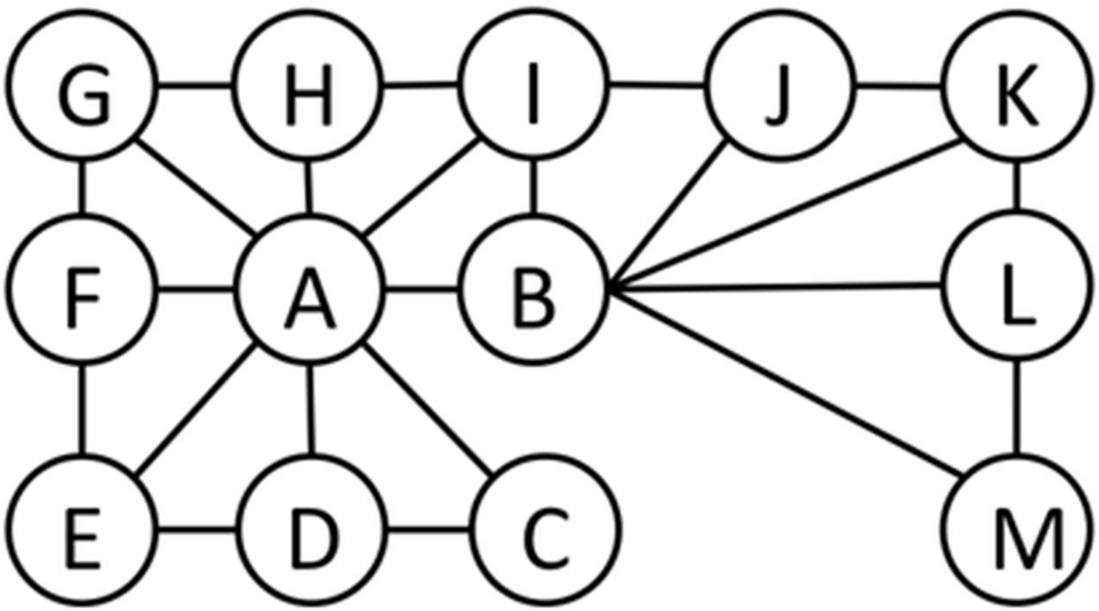
ompute residual CSP
for each assignment

olve the residual CSPs
(tree structured)

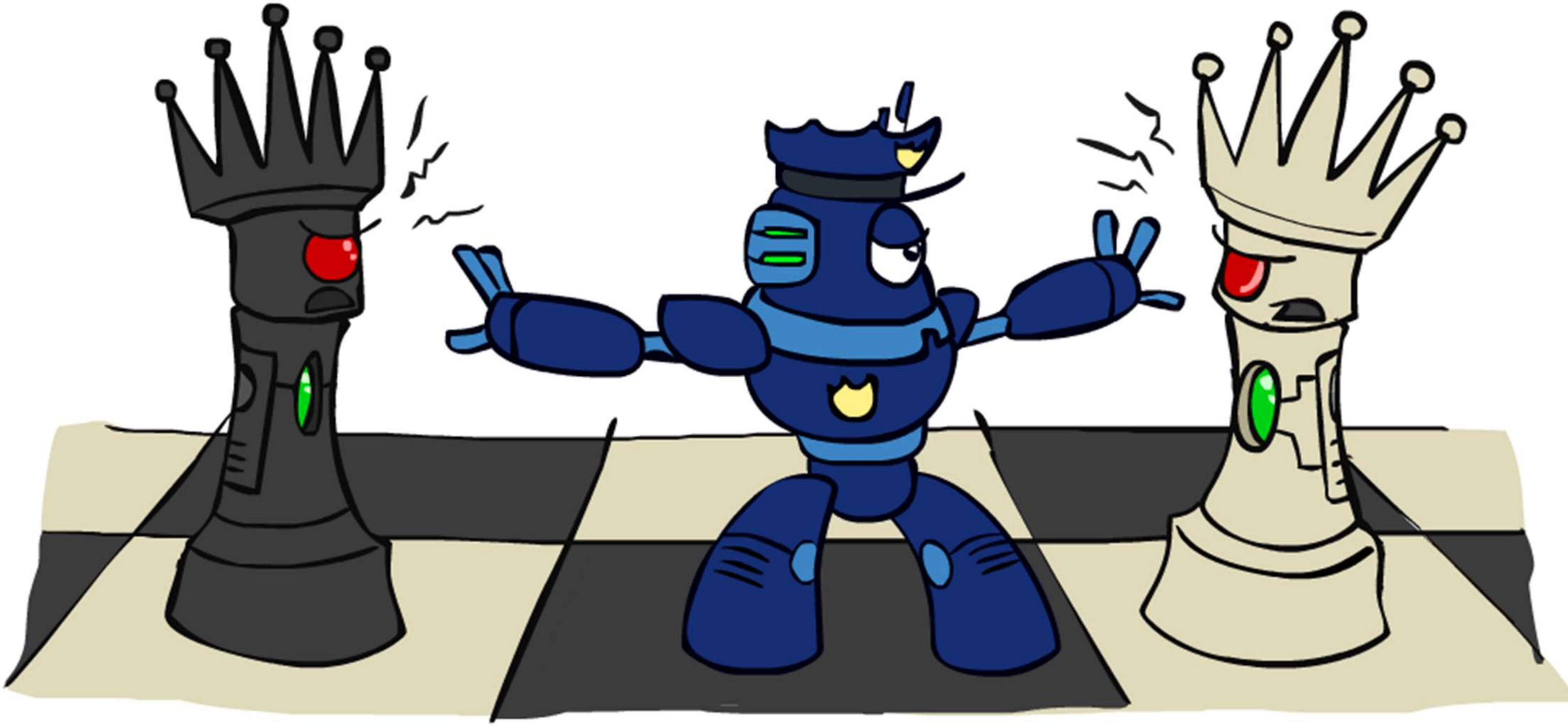


Cutset Quiz

ind the smallest cutset for the graph below.



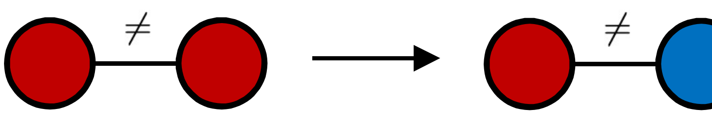
Local Search for CSPs



Iterative Algorithms for CSPs

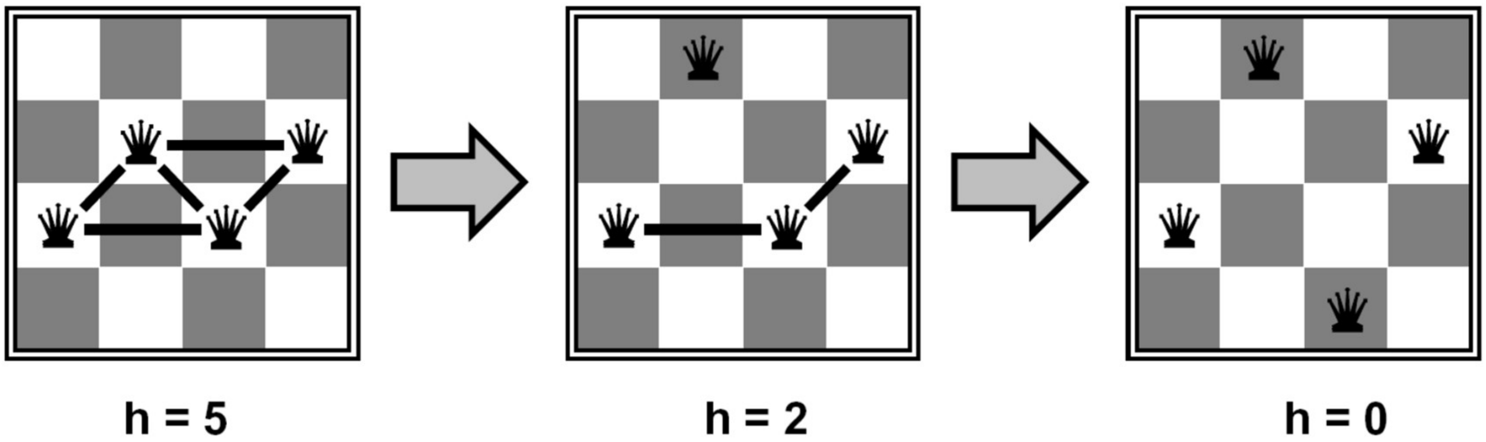
- ocal search methods typically work with "complete" states, i.e., all variables assign

- o apply to CSPs:
  - Take an assignment with unsatisfied constraints
  - Operators *reassign* variable values
  - No fringe! Live on the edge.



- gorithm: 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

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

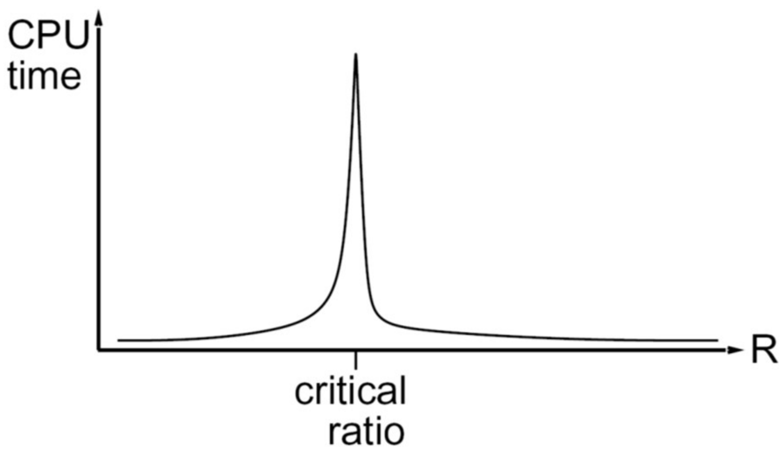
[Demo: n-queens – iterative improveme
[Demo: coloring – iterative improvemen

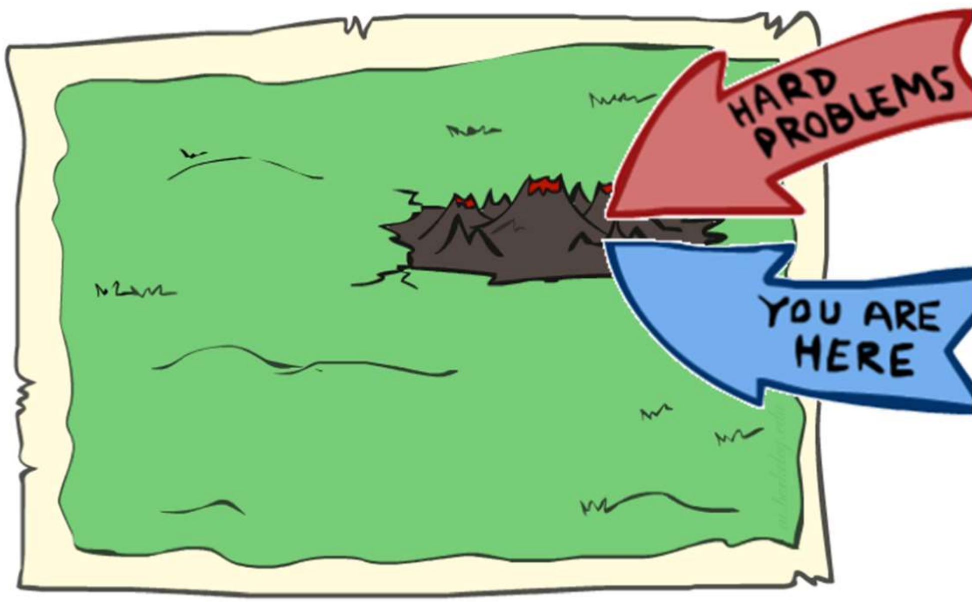
Performance of Min-Conflicts

iven random initial state, can solve n-queens in almost constant time for arbitrary
with high probability (e.g., n = 10,000,000)!

he same appears to be true for any randomly-generated CSP *except* in a narrow
ange of the ratio

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





Summary: CSPs

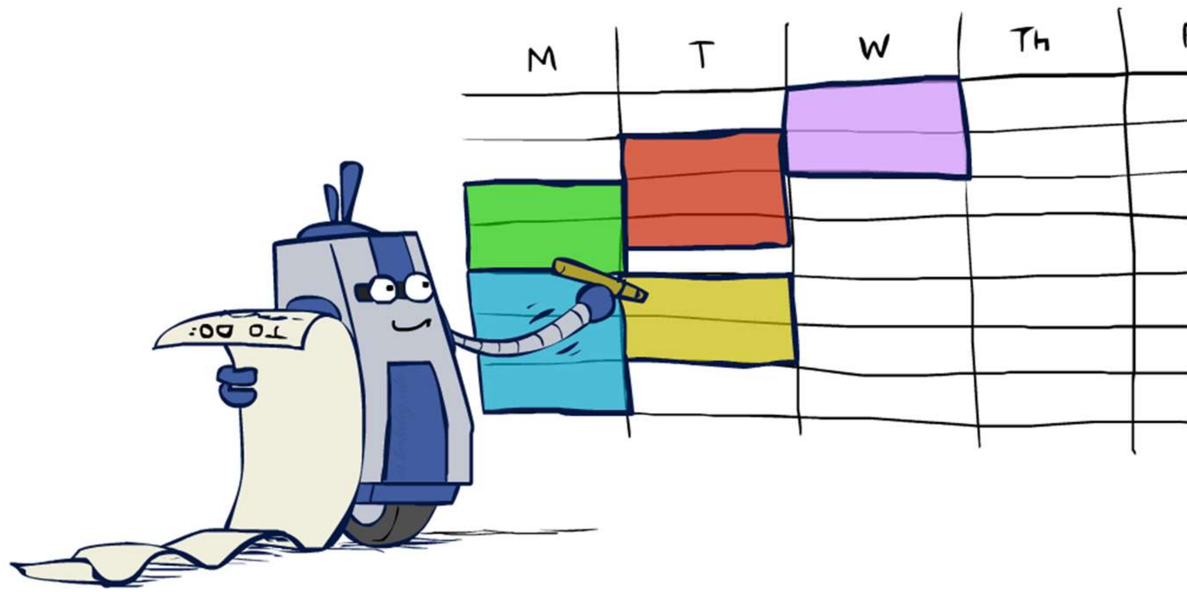
SPs are a special kind of search problem:

- States are partial assignments
- Goal test defined by constrai

asic solution: backtracking sea

peed-ups:

- Ordering
- Filtering
- Structure



terative min-conflicts is often effective in practice