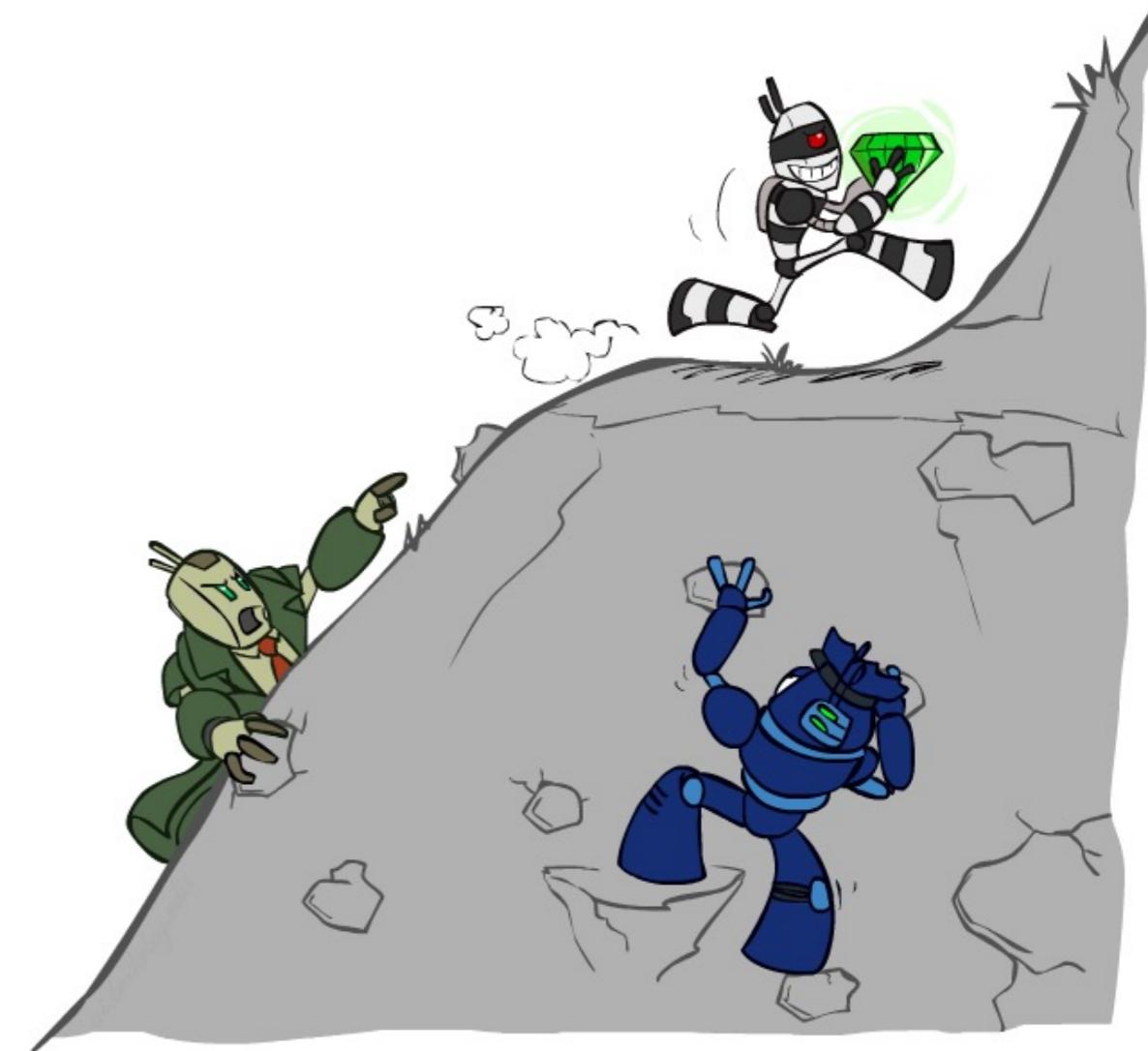


Ve492: Introduction to Artificial Intelligence

Constraint Satisfaction Problems II and Local Search



Paul Weng

UM-SJTU Joint Institute

Slides adapted from <http://ai.berkeley.edu>, AIMA, UM, CMU

Today

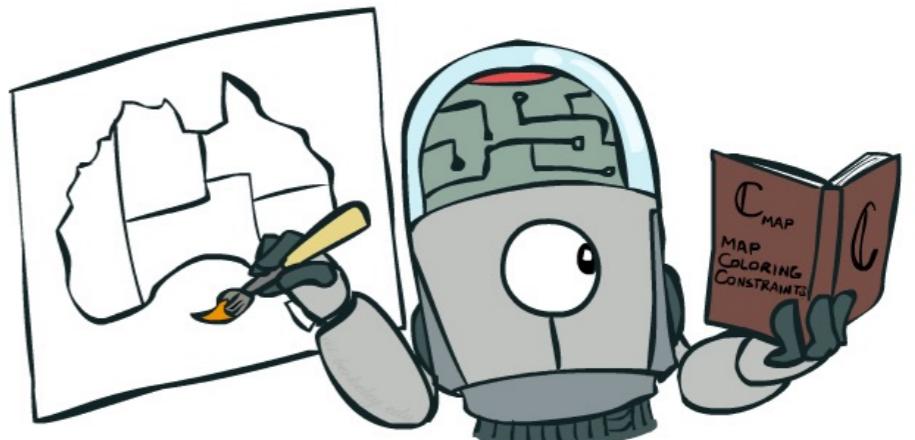
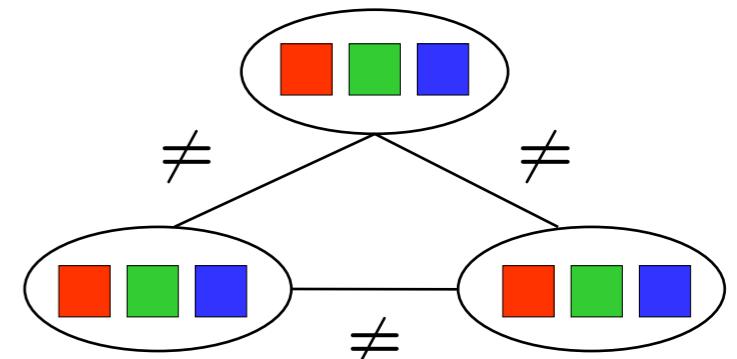
- ❖ Efficient Solution of CSPs

- ❖ Local Search



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  $\leftarrow$  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  $\leftarrow$  RECURSIVE-BACKTRACKING(assignment, csp)
      if result  $\neq$  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?

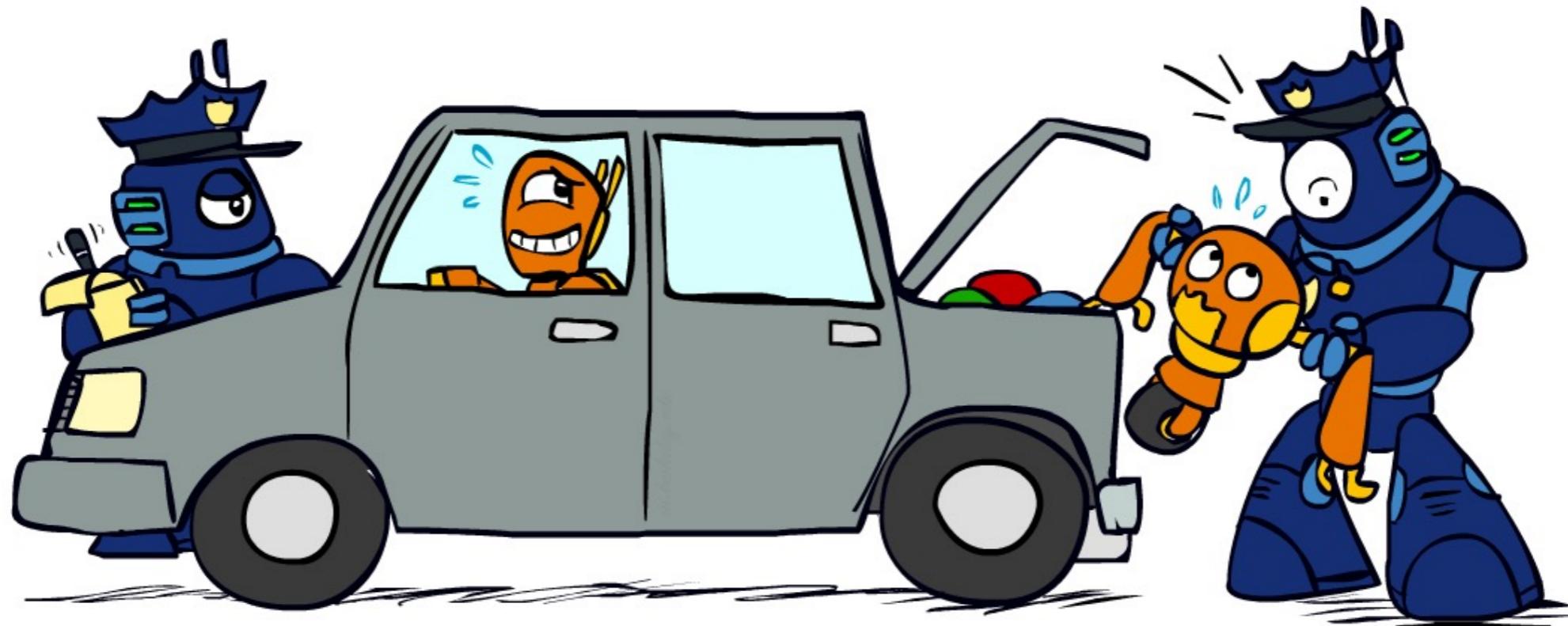
- ❖ Ordering:

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



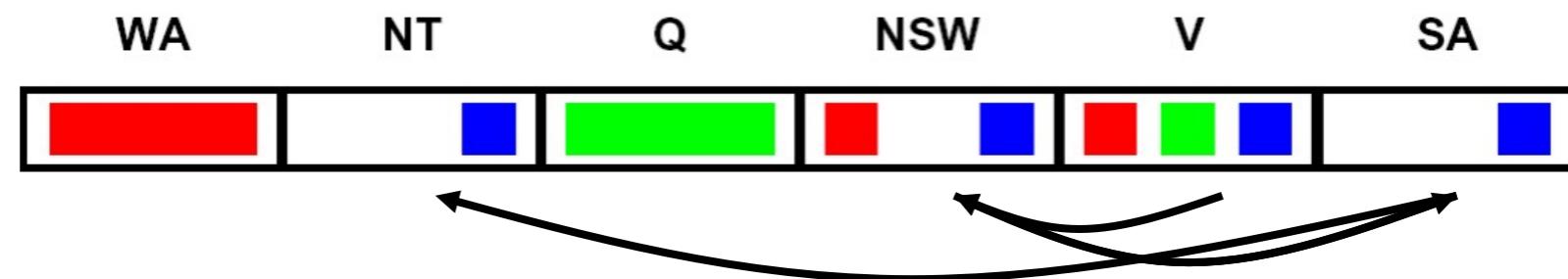
- ❖ Structure: Can we exploit the problem structure?

Arc Consistency and Beyond



Arc Consistency of an Entire CSP

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

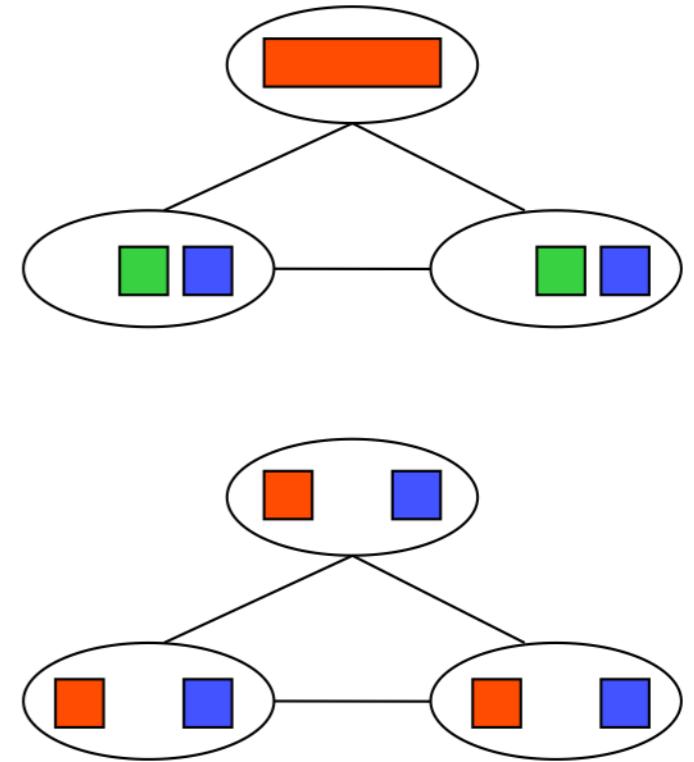


*Remember:
Delete from
the tail!*

- ❖ 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!

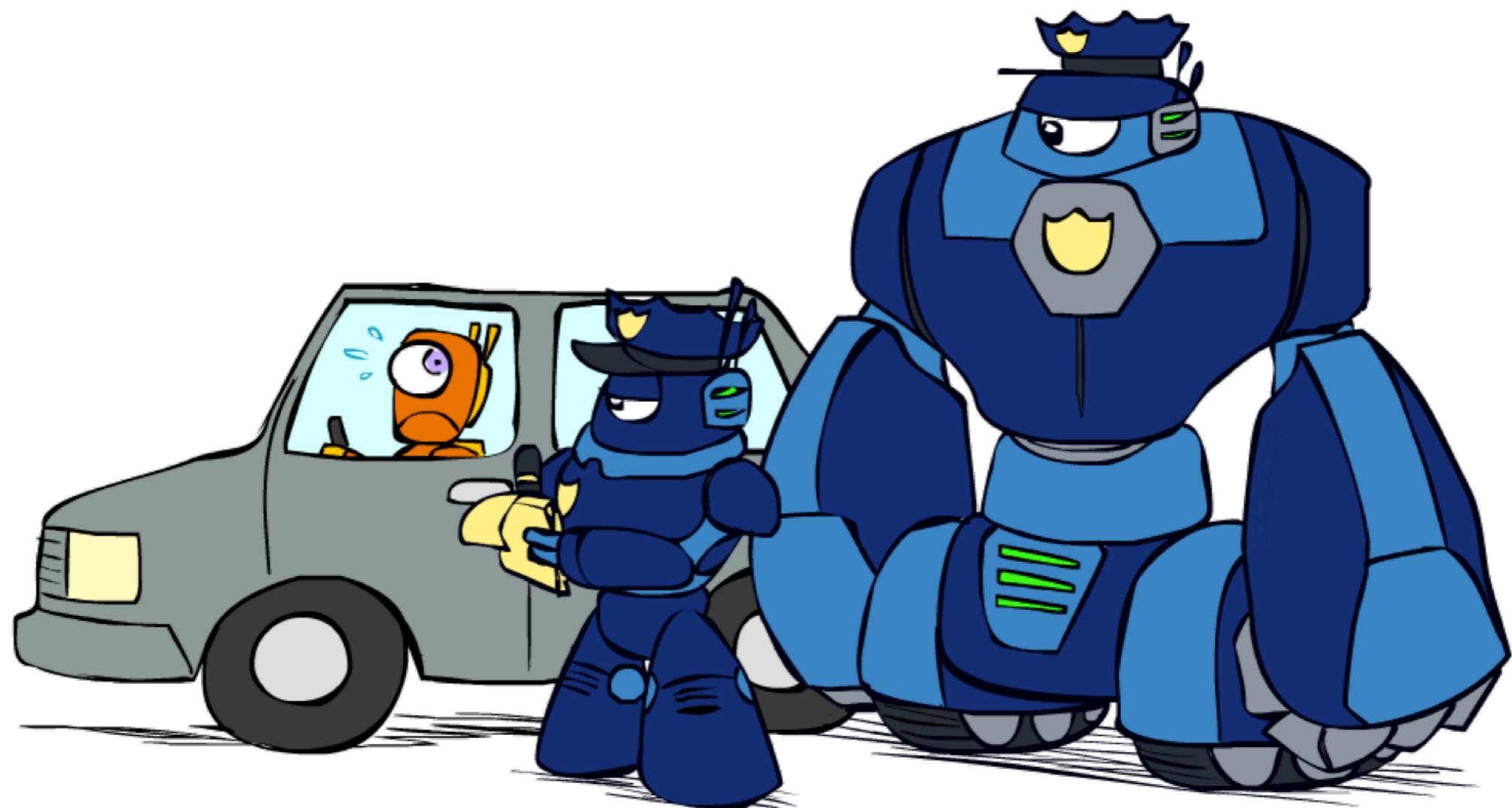
Limitations of Arc Consistency

- ❖ 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?

K-Consistency



K-Consistency

❖ Increasing degrees of consistency

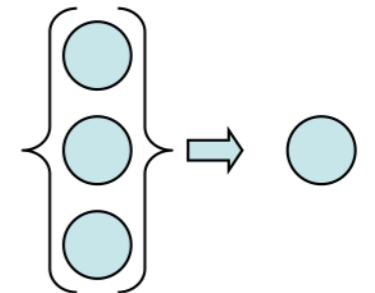
- ❖ 1-Consistency (Node Consistency): Each single node's domain has a value which meets that node's unary constraints



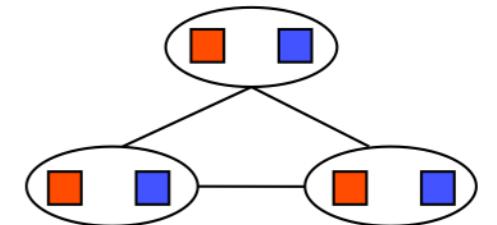
- ❖ 2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other



- ❖ K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the kth node.



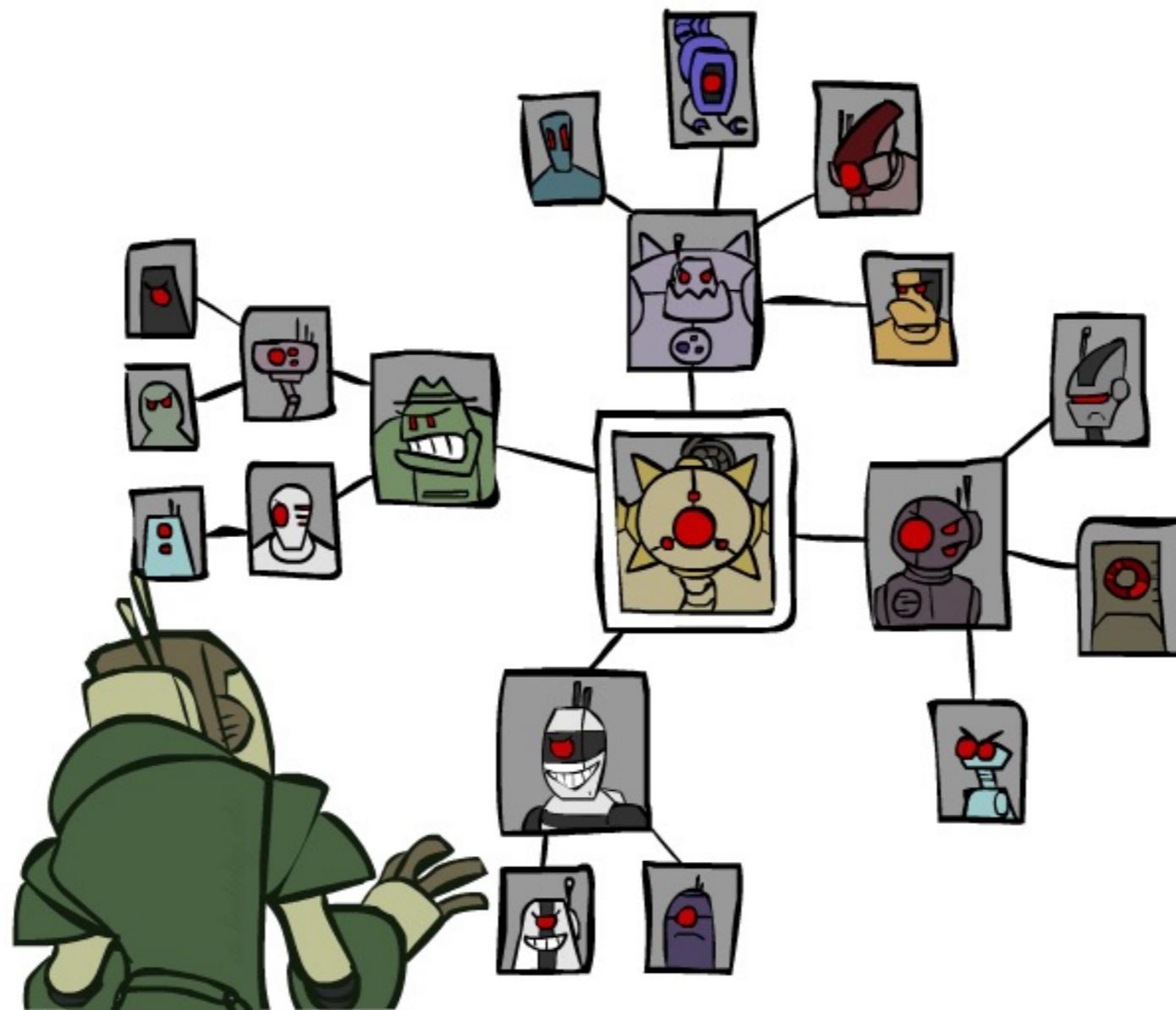
- ❖ Higher k more expensive to compute



Strong K-Consistency

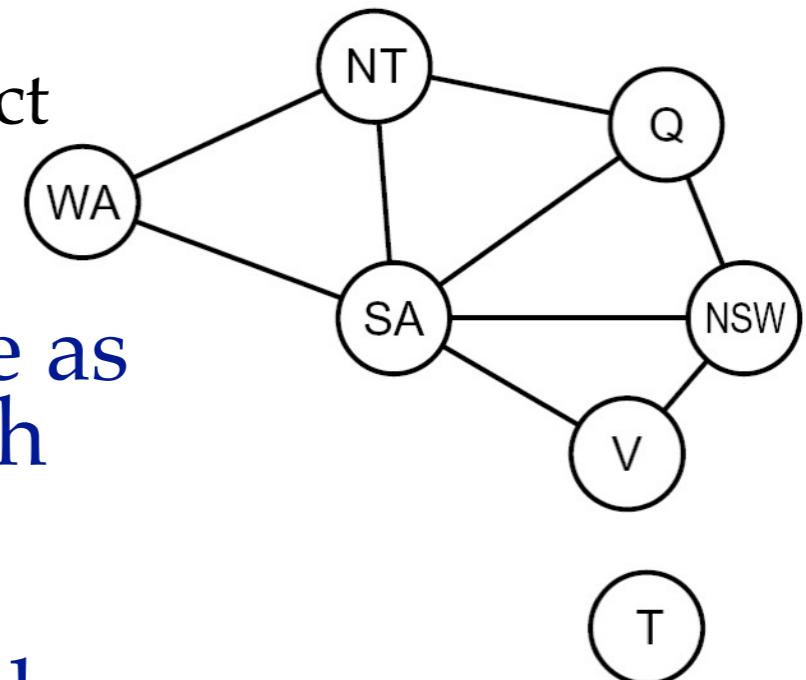
- ❖ Strong k-consistency: also k-1, k-2, ... 1 consistent
- ❖ Claim: strong n-consistency means we can solve without backtracking!
- ❖ Why?
 - ❖ Choose any assignment to any variable
 - ❖ Choose a new variable
 - ❖ By 2-consistency, there is a choice consistent with the first
 - ❖ Choose a new variable
 - ❖ 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)

Structure

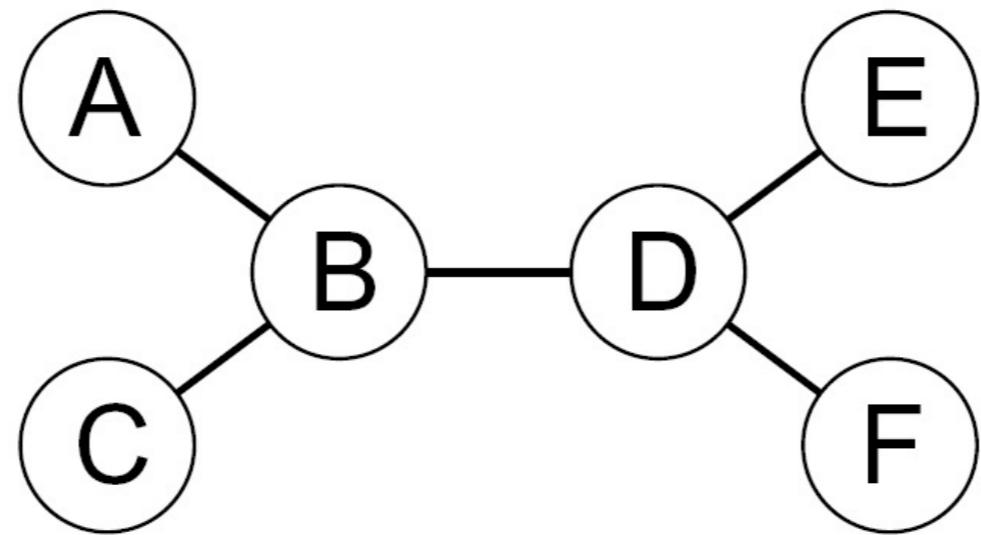


Problem Structure

- ❖ 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., $n = 80$, $d = 2$, $c = 20$
 - ❖ $2^{80} = 4 \text{ billion years}$ at 10 million nodes/sec
 - ❖ $(4)(2^{20}) = 0.4 \text{ seconds}$ at 10 million nodes/sec



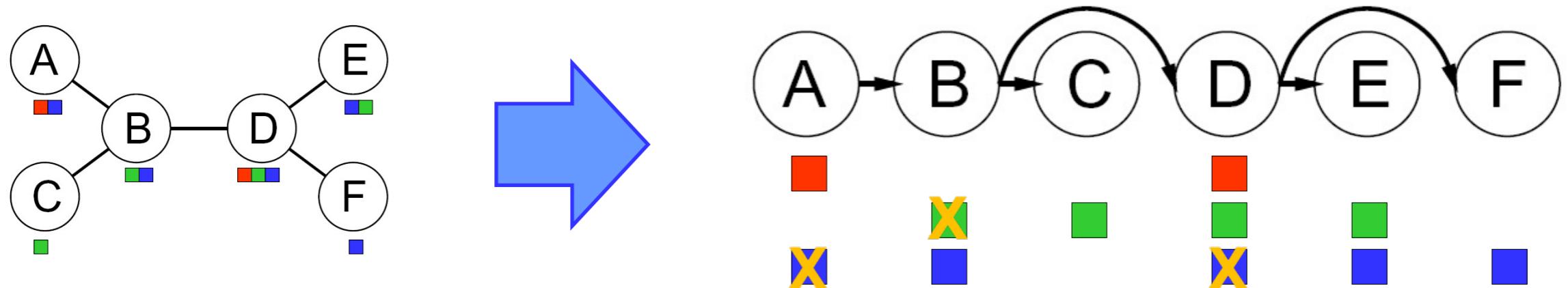
Tree-Structured CSPs



- ❖ 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

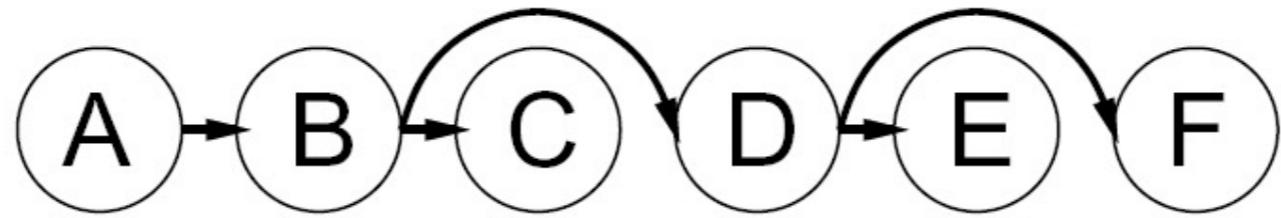


- ❖ Remove backward: For $i = n : 2$, apply $\text{RemoveInconsistent}(\text{Parent}(X_i), X_i)$
- ❖ Assign forward: For $i = 1 : n$, assign X_i consistently with $\text{Parent}(X_i)$
- ❖ Runtime: $O(n d^2)$ (why?)



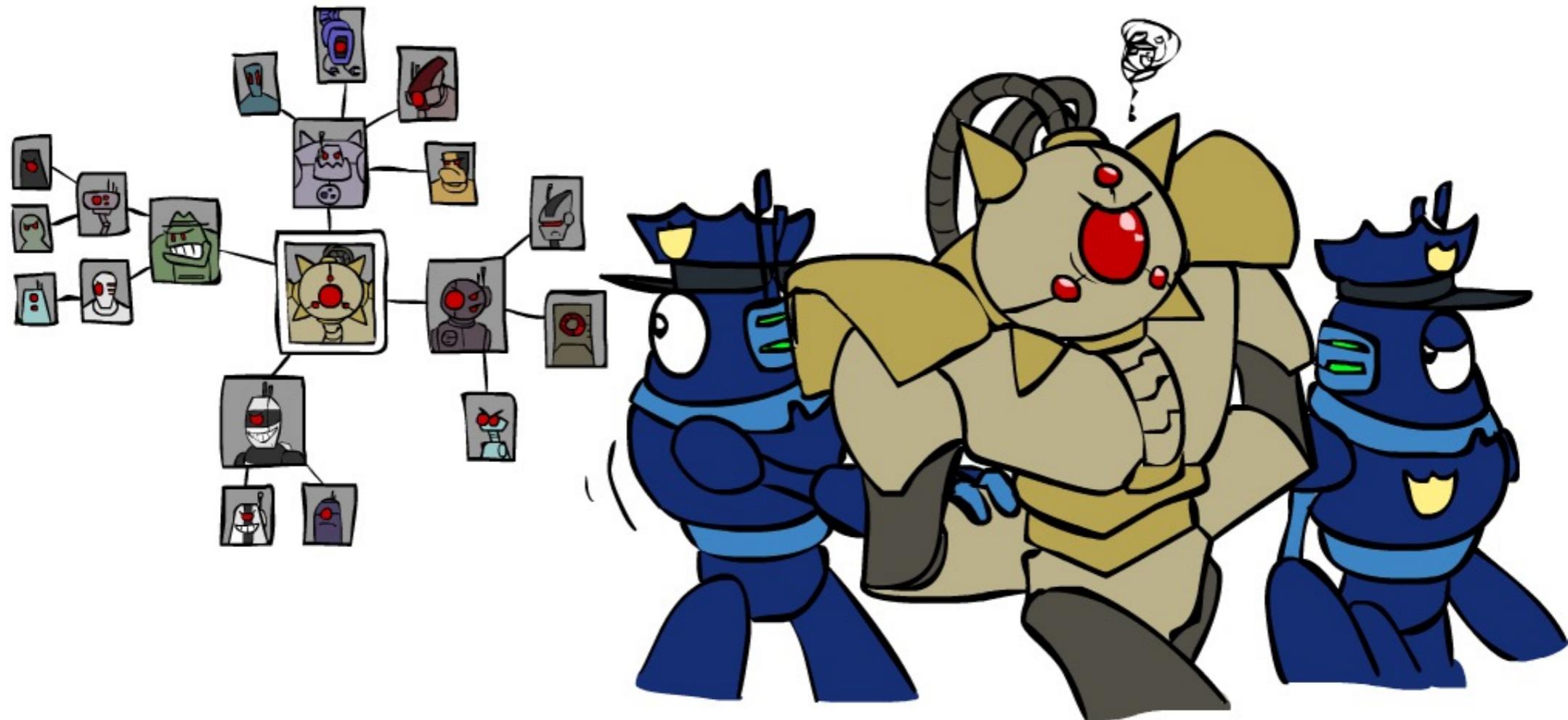
Tree-Structured CSPs

- ❖ 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)

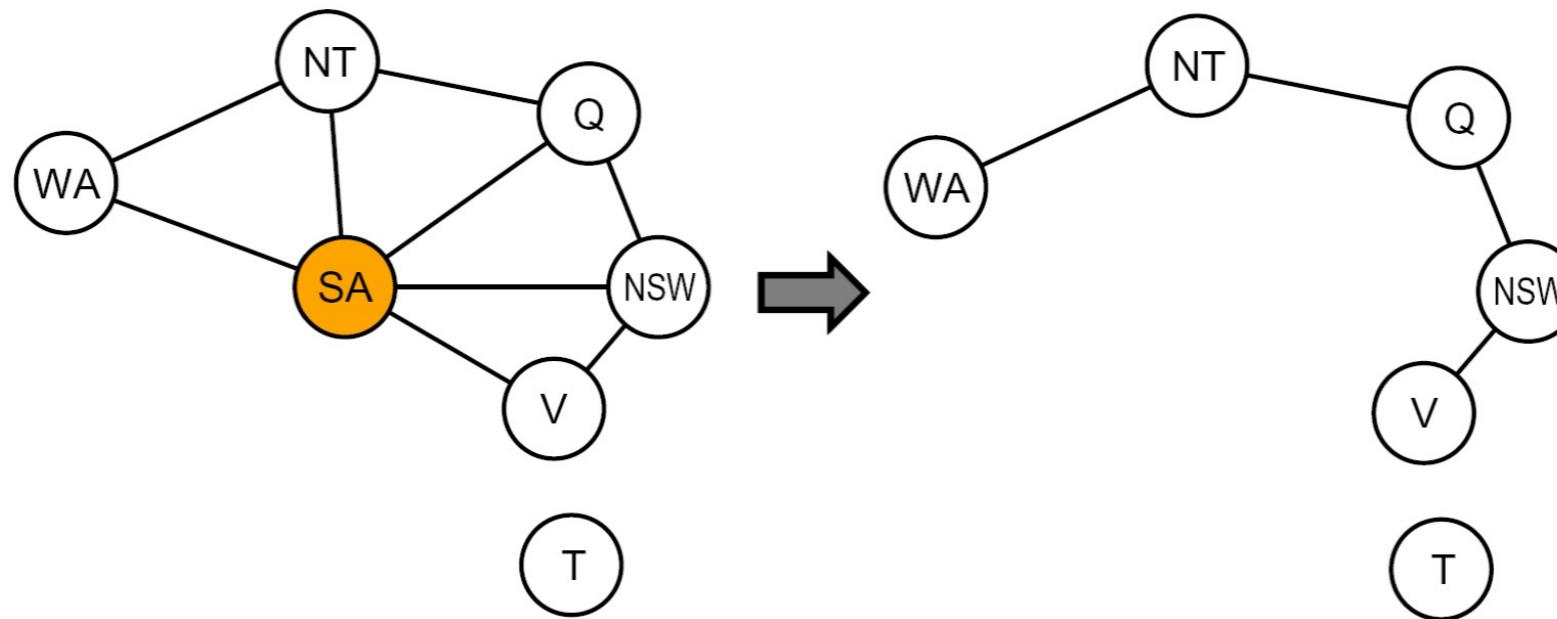


- ❖ Claim 2: If root-to-leaf arcs are consistent, forward assignment will not backtrack
- ❖ Proof: Induction on position
- ❖ Why doesn't this algorithm work with cycles in the constraint graph?
- ❖ Note: we'll see this basic idea again with Bayes' nets

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

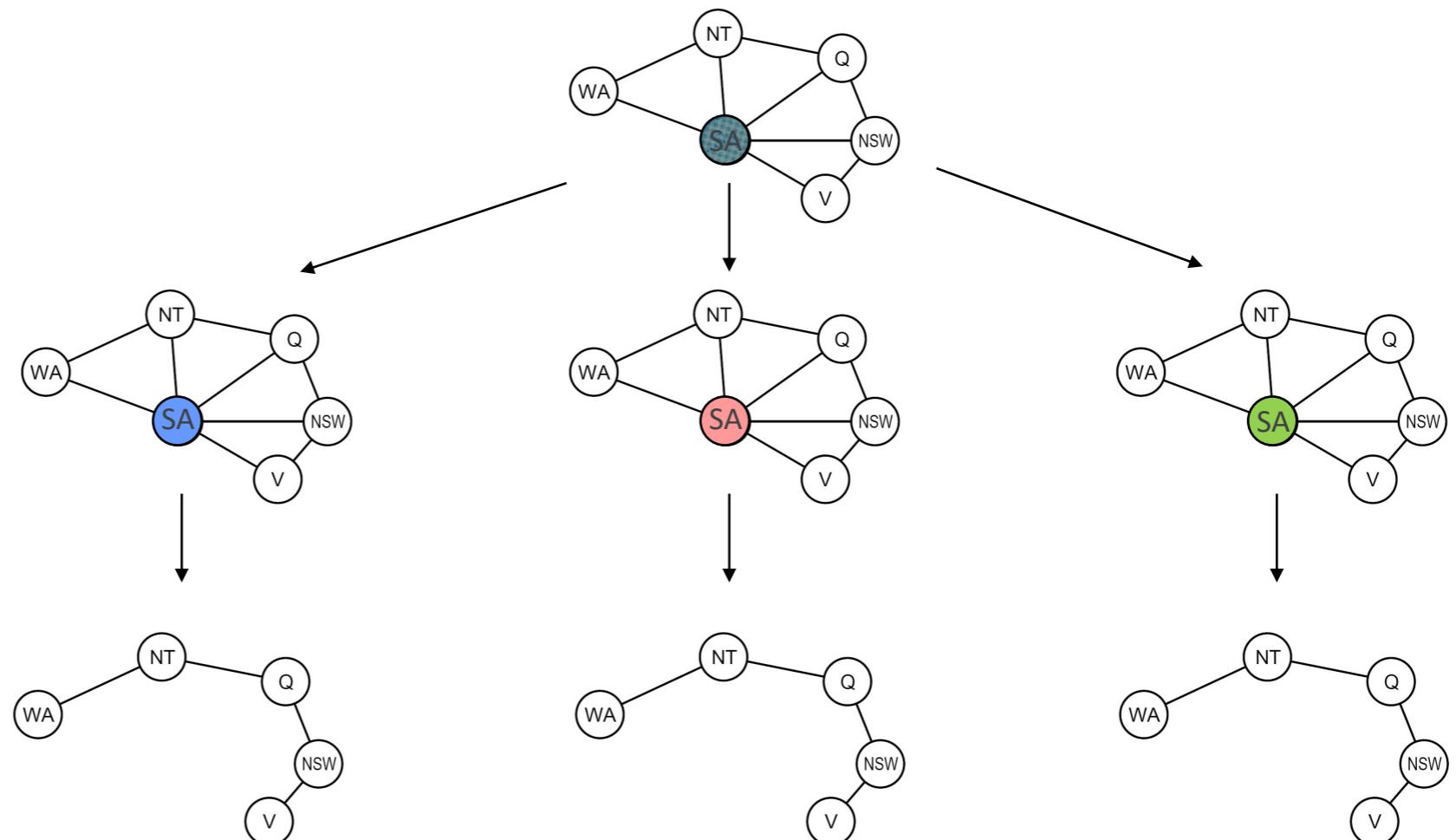
Cutset Conditioning

Choose a cutset

Instantiate the cutset
(all possible ways)

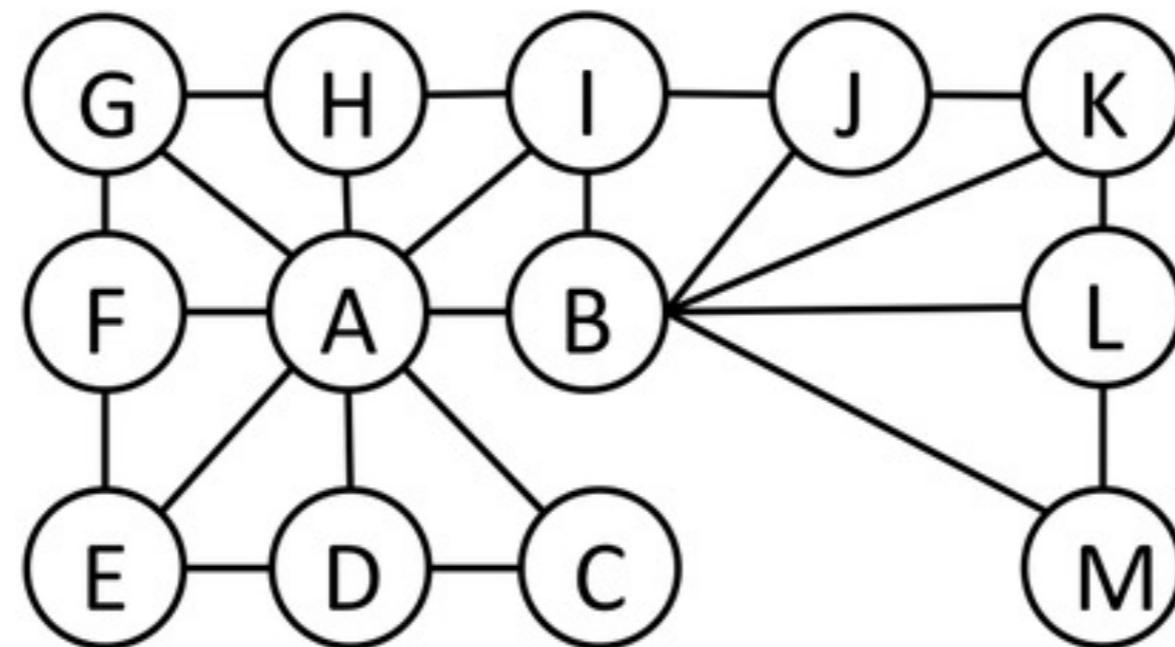
Compute residual CSP
for each assignment

Solve the residual
CSPs (tree structured)



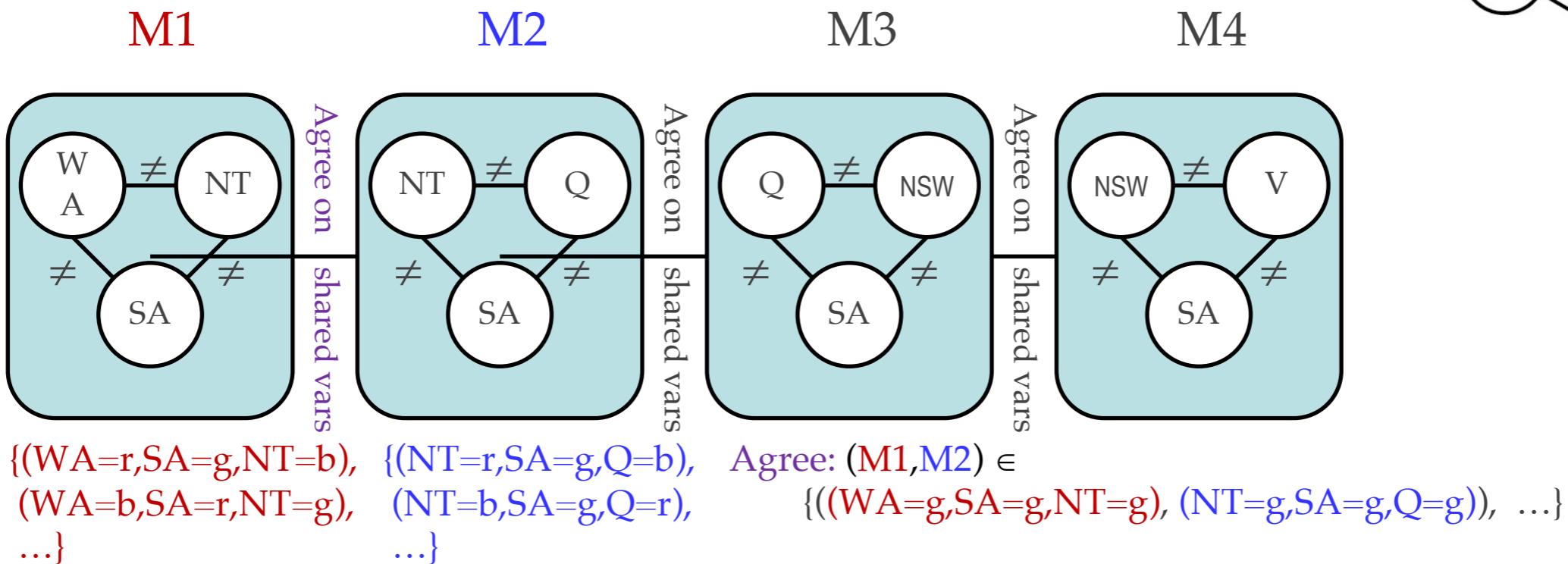
Quiz: Cutset

- ❖ Find the smallest cutset for the graph below.

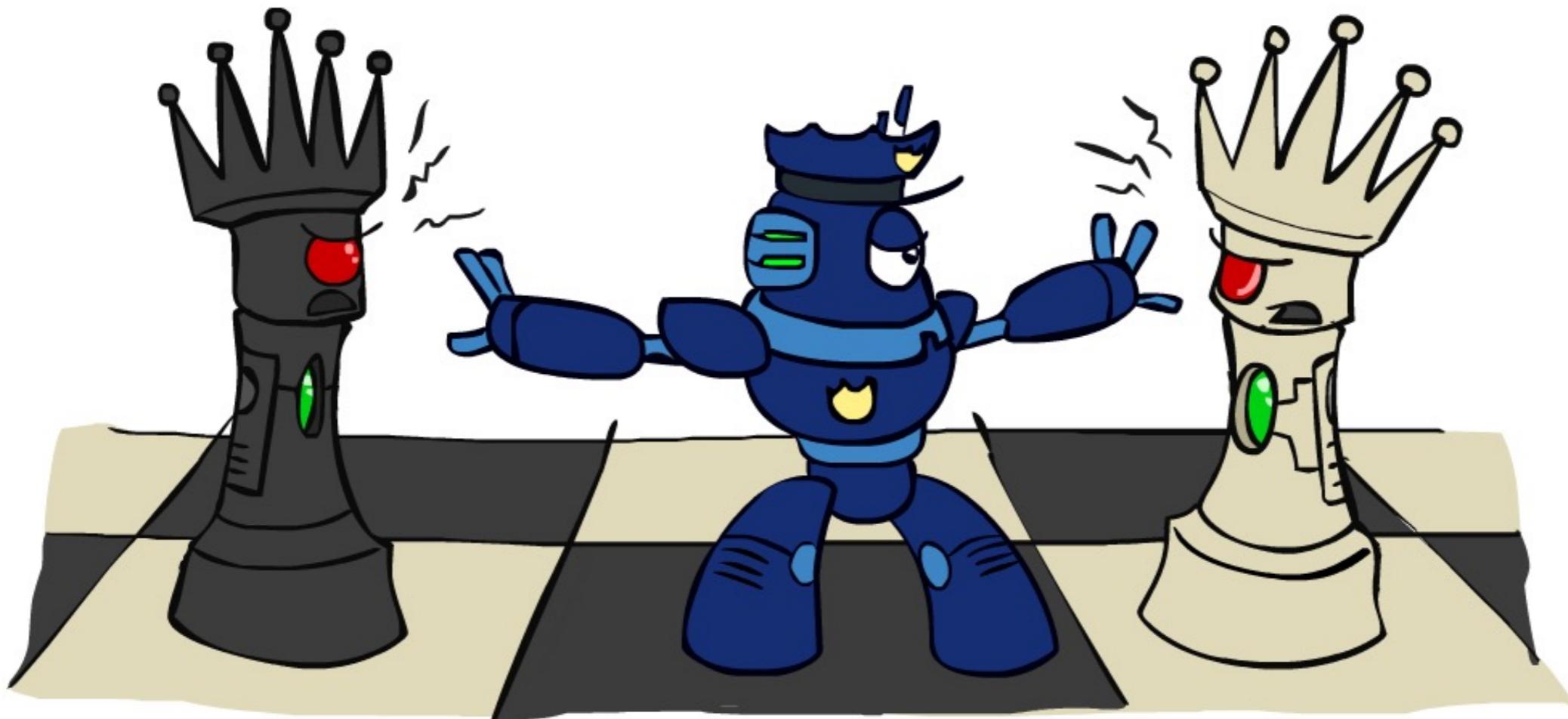


Tree Decomposition *

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



Iterative Improvement

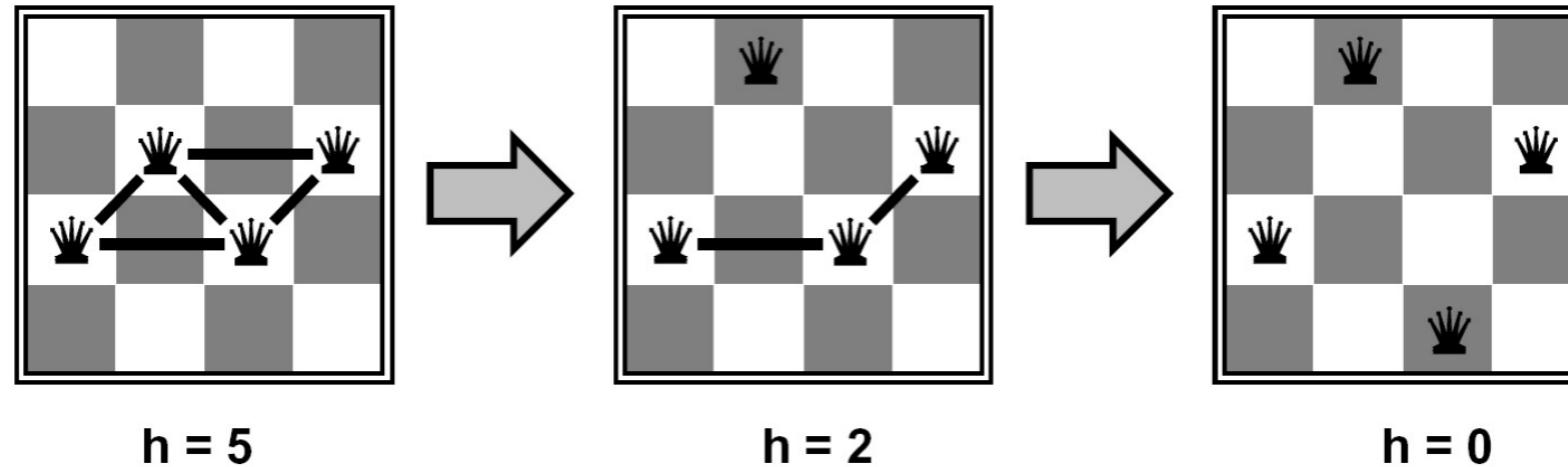


Iterative Algorithms for CSPs

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

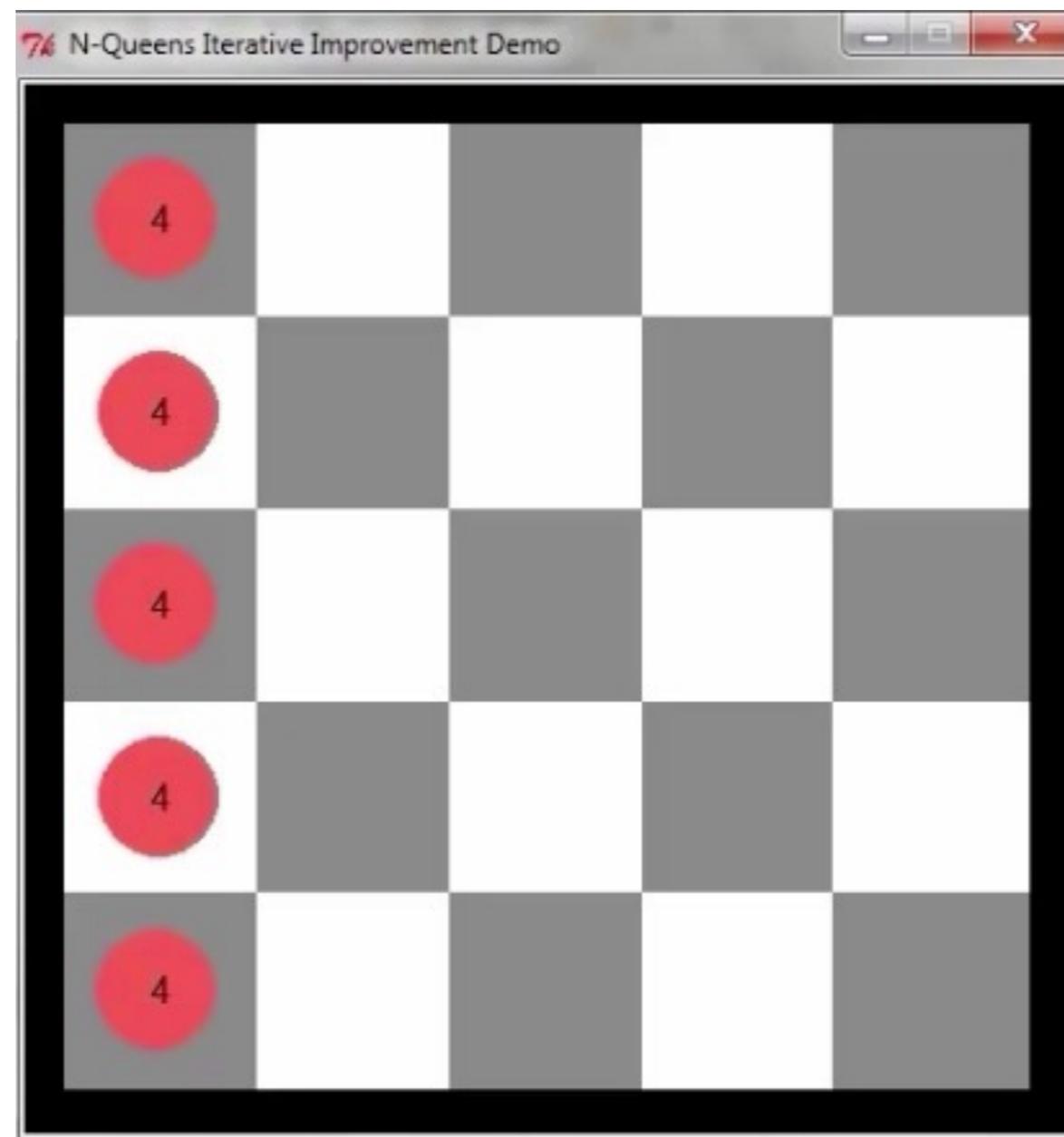


Example: 4-Queens

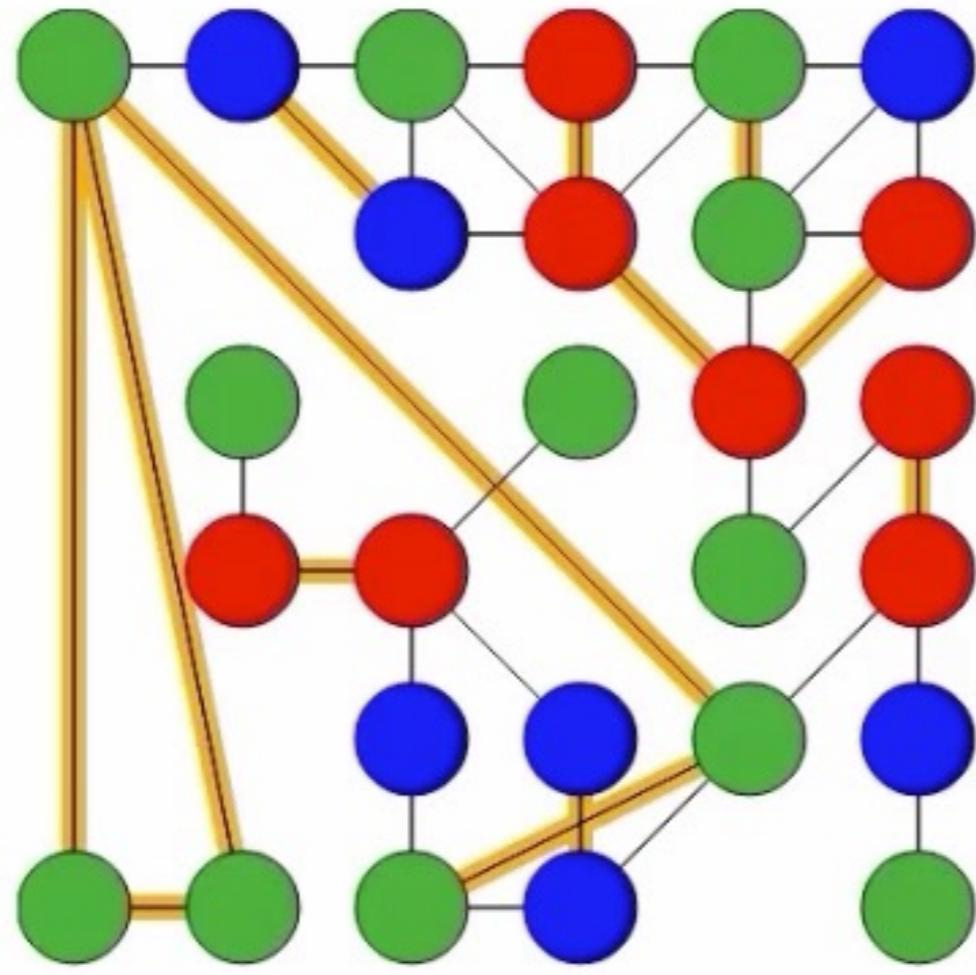


- ❖ States: 4 queens in 4 columns ($4^4 = 256$ states)
- ❖ Operators: move queen in column
- ❖ Goal test: no attacks
- ❖ Evaluation: $h(n)$ = number of attacks

Video of Demo Iterative Improvement – n Queens



Video of Demo Iterative Improvement – Coloring



Reset Prev Pause Next Play Faster

Graph

Complex

Algorithm

Iterative Improvement

Ordering

- None
- MRV
- MRV with LCV

Filtering

- None
- Forward Checking
- Arc Consistency

Speed

Speedup

1 x

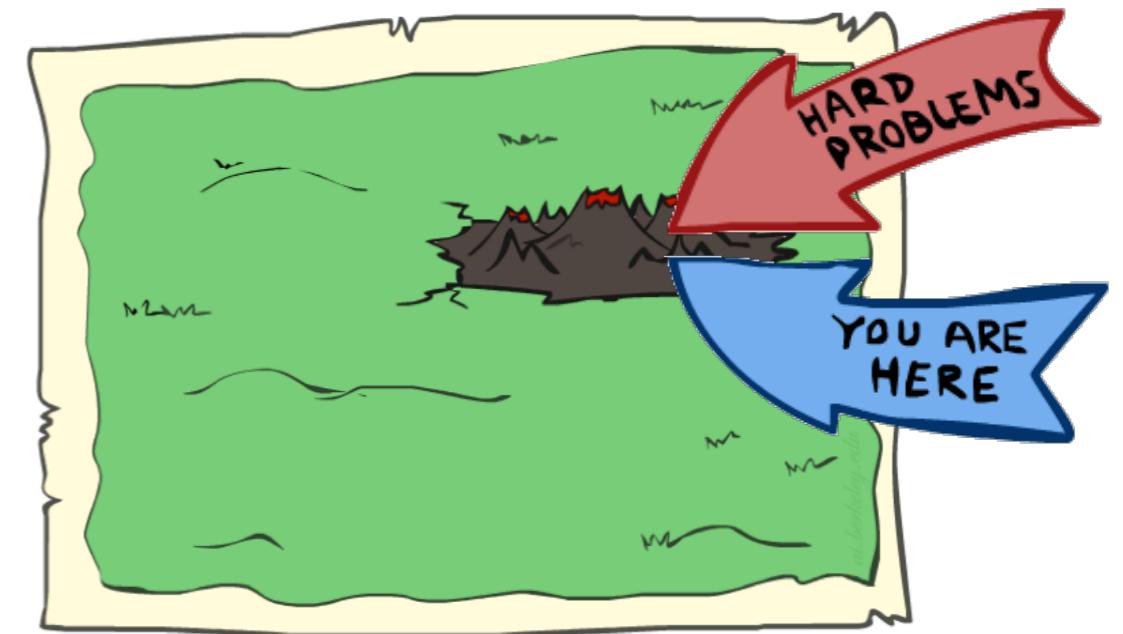
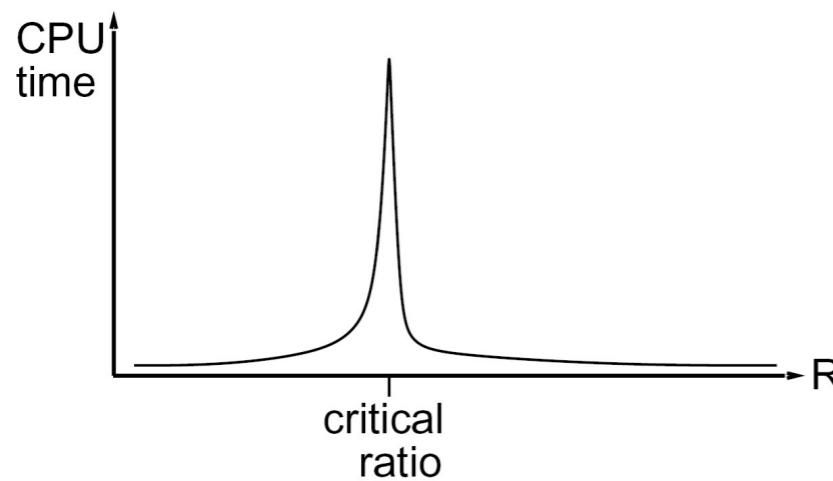
Frame Delay

700

Performance of Min-Conflicts

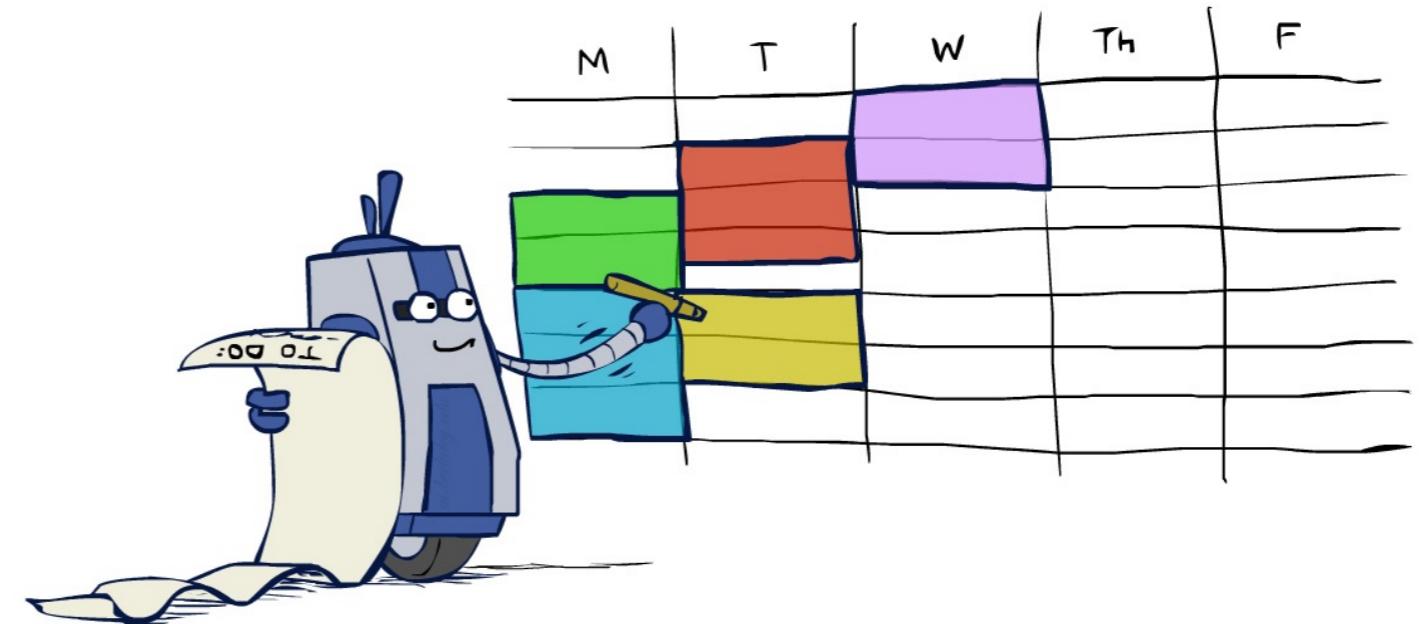
- ❖ Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)!
- ❖ The same appears to be true for any randomly-generated CSP *except* in a narrow range of the ratio

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



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

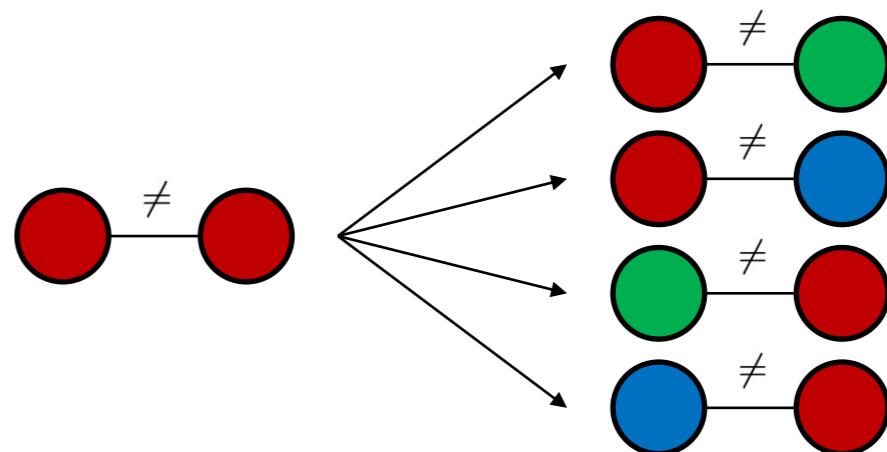


Local Search



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)

Hill Climbing

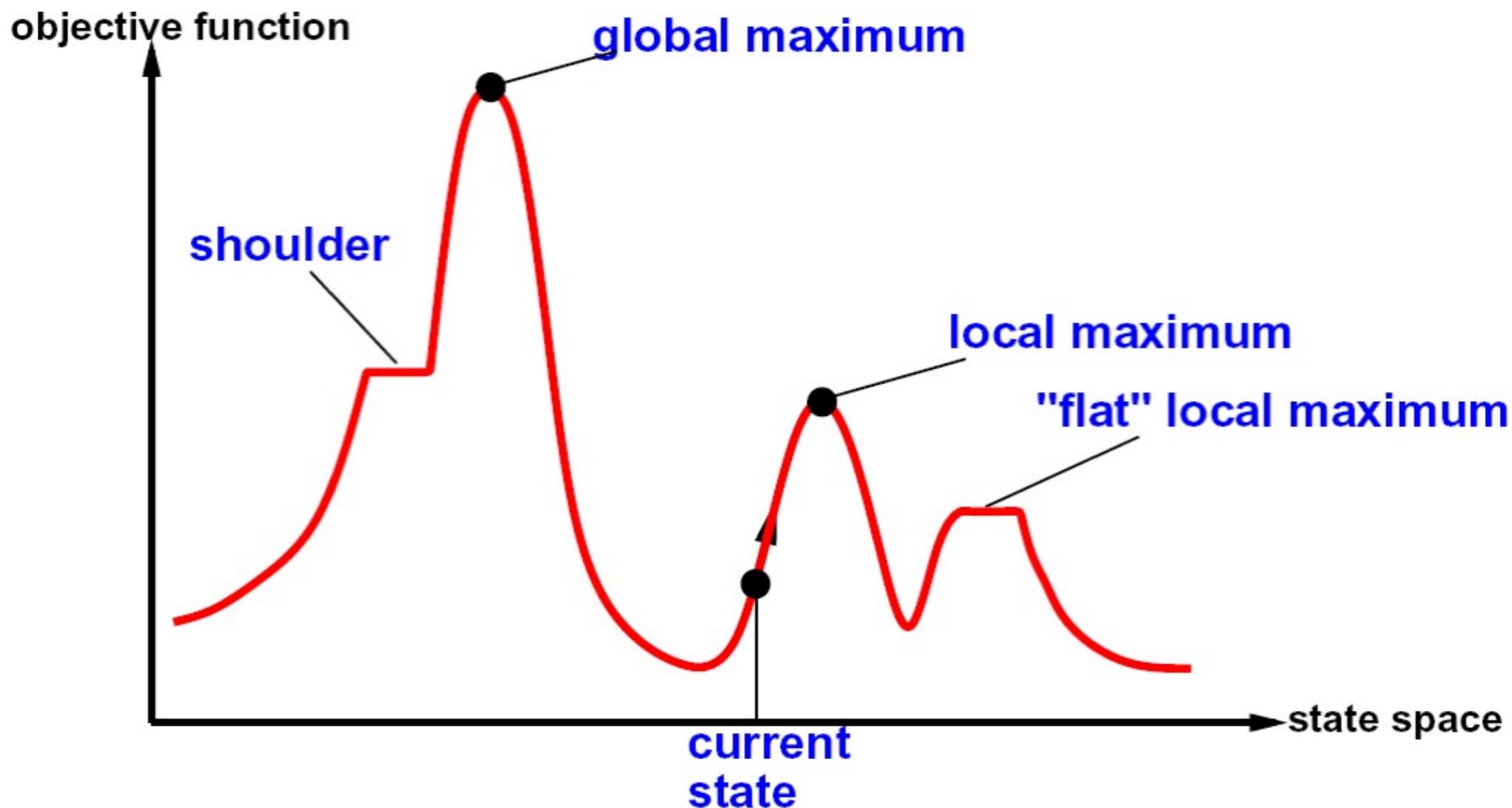
- ❖ Simple, general idea:
 - ❖ Start wherever
 - ❖ Repeat: move to the best neighboring state
 - ❖ If no neighbors better than current, quit
- ❖ What's bad about this approach?
 - ❖ Complete?
 - ❖ Optimal?
- ❖ What's good about it?



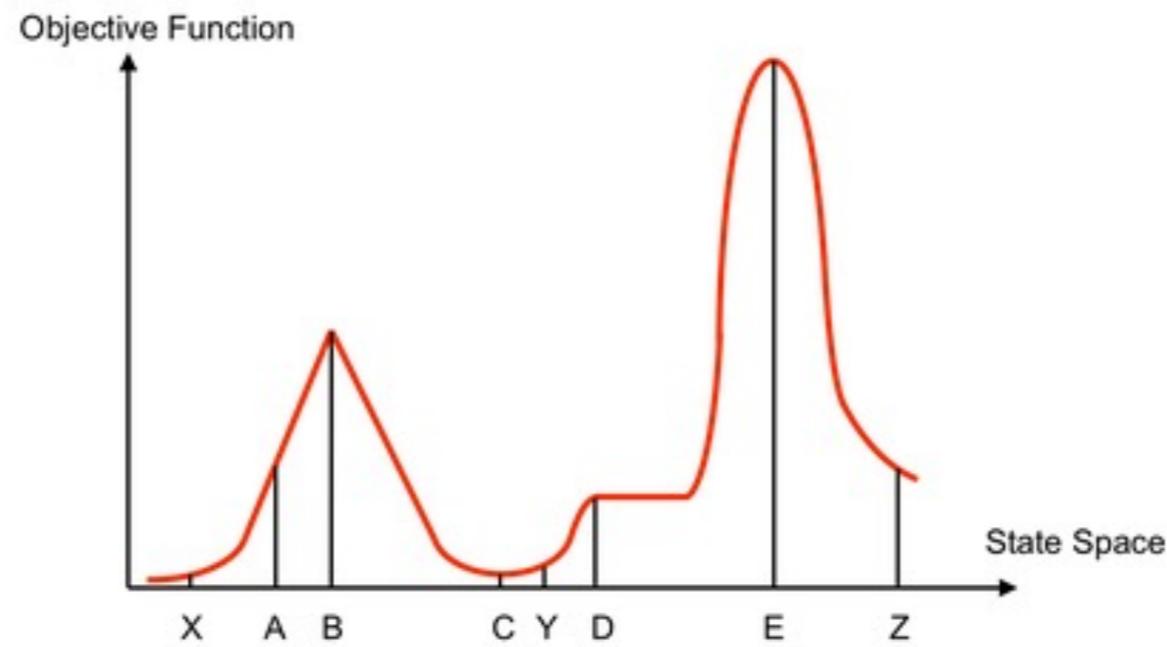
Hill Climbing Algorithm

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                  neighbor, a node
  current  $\leftarrow$  MAKE-NODE(INITIAL-STATE[problem])
  loop do
    neighbor  $\leftarrow$  a highest-valued successor of current
    if VALUE[neighbor]  $\leq$  VALUE[current] then return STATE[current]
    current  $\leftarrow$  neighbor
  end
```

Hill Climbing Diagram



Quiz: Hill Climbing



Starting from X, where do you end up ? 

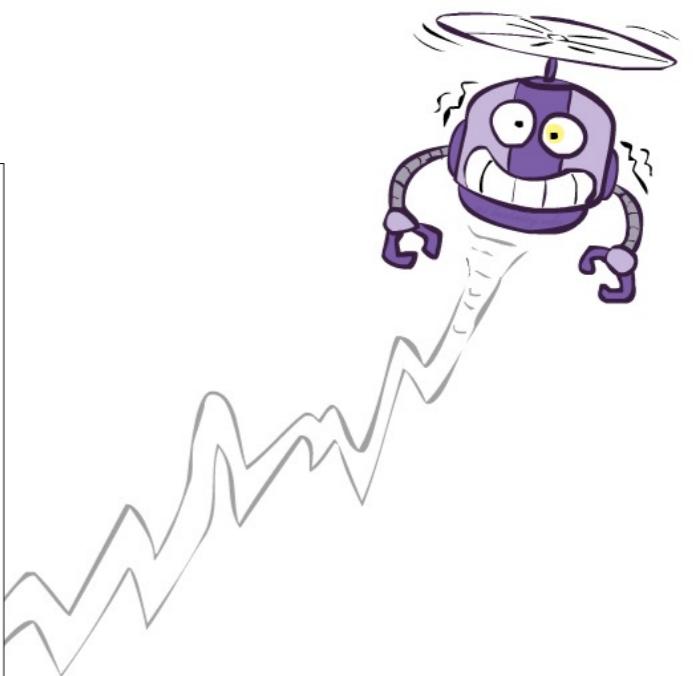
Starting from Y, where do you end up ?

Starting from Z, where do you end up ?

Simulated Annealing

- ❖ Idea: Escape local maxima by allowing downhill moves
 - ❖ But make them rarer as time goes on

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to “temperature”
  local variables: current, a node
                    next, a node
                    T, a “temperature” controlling prob. of downward steps
  current  $\leftarrow$  MAKE-NODE(INITIAL-STATE[problem])
  for t  $\leftarrow$  1 to  $\infty$  do
    T  $\leftarrow$  schedule[t]
    if T = 0 then return current
    next  $\leftarrow$  a randomly selected successor of current
     $\Delta E \leftarrow$  VALUE[next] – VALUE[current]
    if  $\Delta E > 0$  then current  $\leftarrow$  next
    else current  $\leftarrow$  next only with probability  $e^{\Delta E/T}$ 
```



Simulated Annealing

- ❖ Theoretical guarantee:

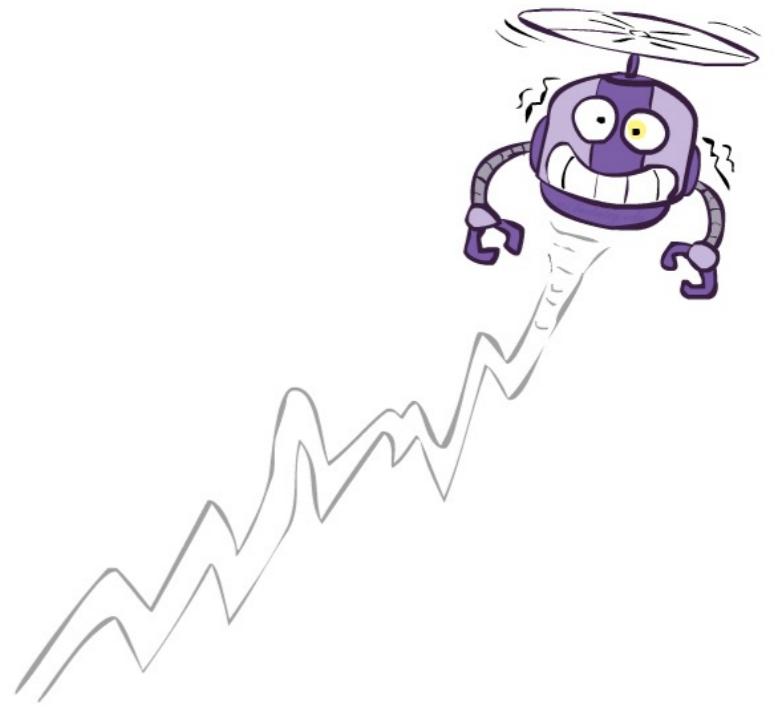
- ❖ Stationary distribution: $p(x) \propto e^{\frac{E(x)}{kT}}$

- ❖ If T decreased slowly enough, will converge to optimal state!

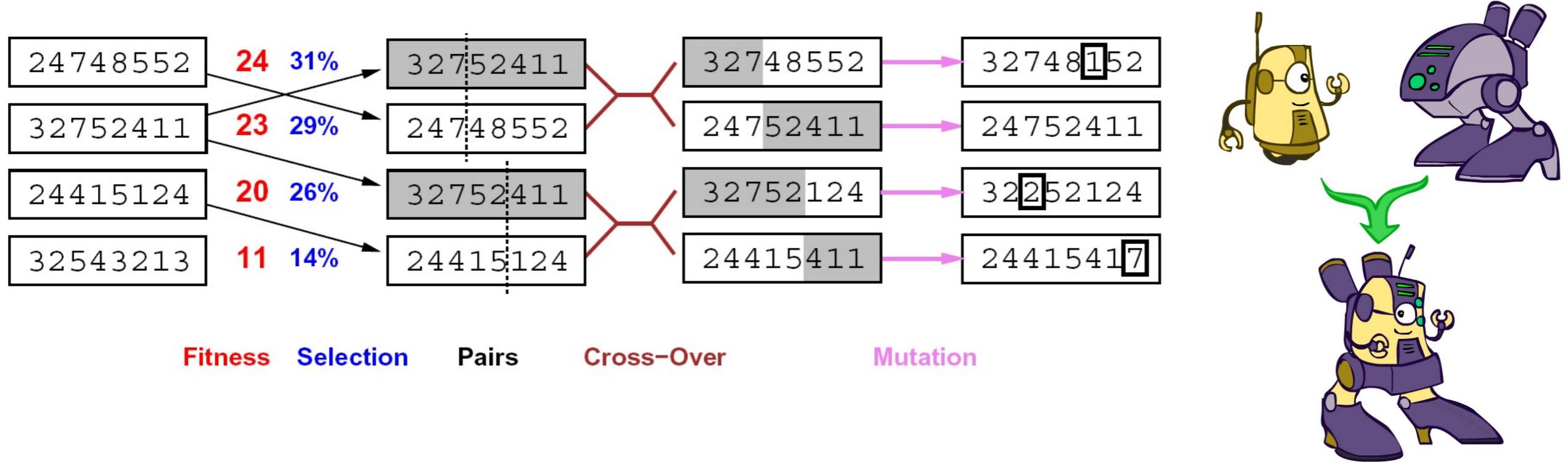
- ❖ Is this an interesting guarantee?

- ❖ Sounds like magic, but reality is reality:

- ❖ The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row
 - ❖ “Slowly enough” may mean exponentially slowly
 - ❖ Random restart hillclimbing also converges to optimal state...

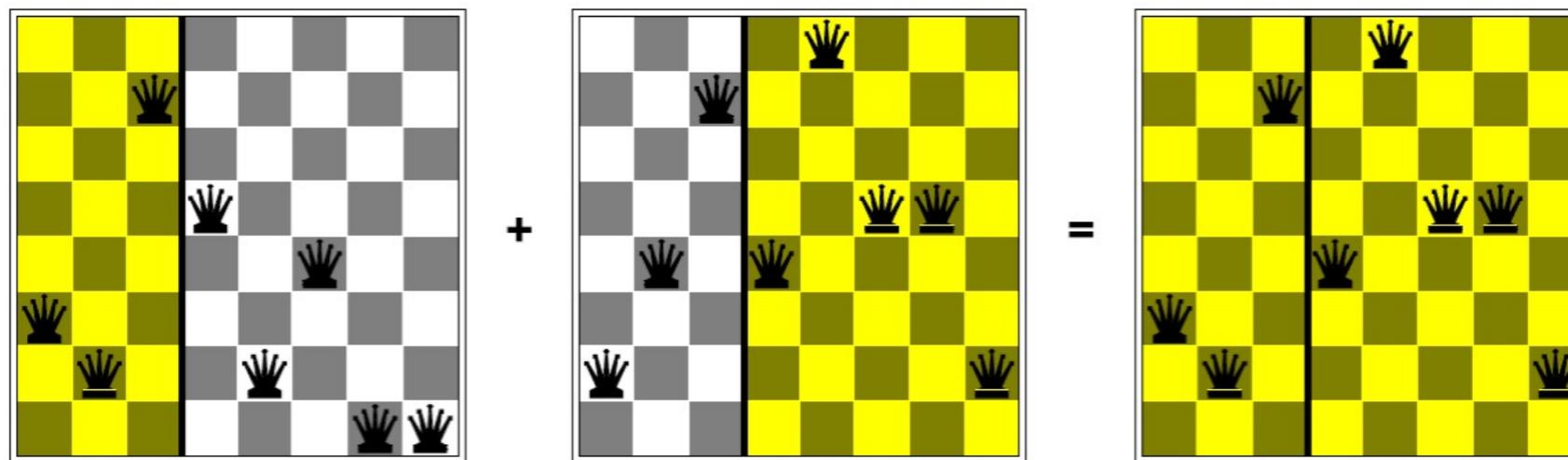


Genetic Algorithms



- ❖ Genetic algorithms use a natural selection metaphor
 - ❖ Keep best N hypotheses at each step (selection) based on a fitness function
 - ❖ Also have pairwise crossover operators, with optional mutation to give variety
- ❖ Possibly the most misunderstood, misapplied (and even maligned) technique around

Example: N-Queens



- ❖ Why does crossover make sense here?
- ❖ When wouldn't it make sense?
- ❖ What would mutation be?
- ❖ What would a good fitness function be?

Mid-term Course Evaluation

