

Announcements

- Project 1 is due **Thursday, April 6th, 11:59 PM PT**
- Please follow the announcements at the Teams chat, the class times will have some changes in the coming weeks

272SM: Introduction to Artificial Intelligence

Constraint Satisfaction Problems II

Instructor: Tatjana Petrov

University of Trieste, Italy



Exercise: Formulating a CSP

Can you phrase the problem of Hamiltonian tour as a CSP (given a network of cities connected by roads, choose an order to visit all cities in a country without repeating any)?

Exercise: Reduction to binary constraints

- *Show how a single ternary constraint such as “ $A+B = C$ ” can be turned into three binary constraints by using an auxiliary variable. You may assume finite domains.*

Today

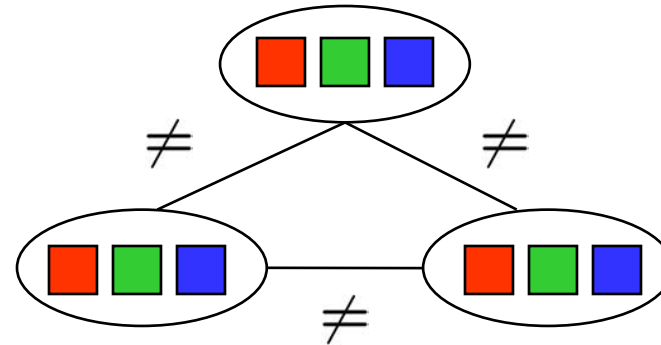
- Efficient Solution of CSPs
- Iterative Improvement



Review: CSPs

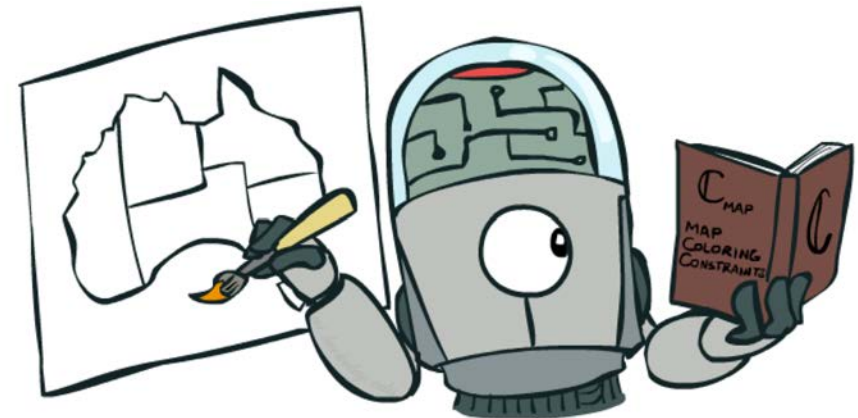
- CSPs:

- Variables
- Domains
- Constraints
 - Implicit (provide code to compute)
 - Explicit (provide a list of the legal tuples)
 - Unary / Binary / N-ary

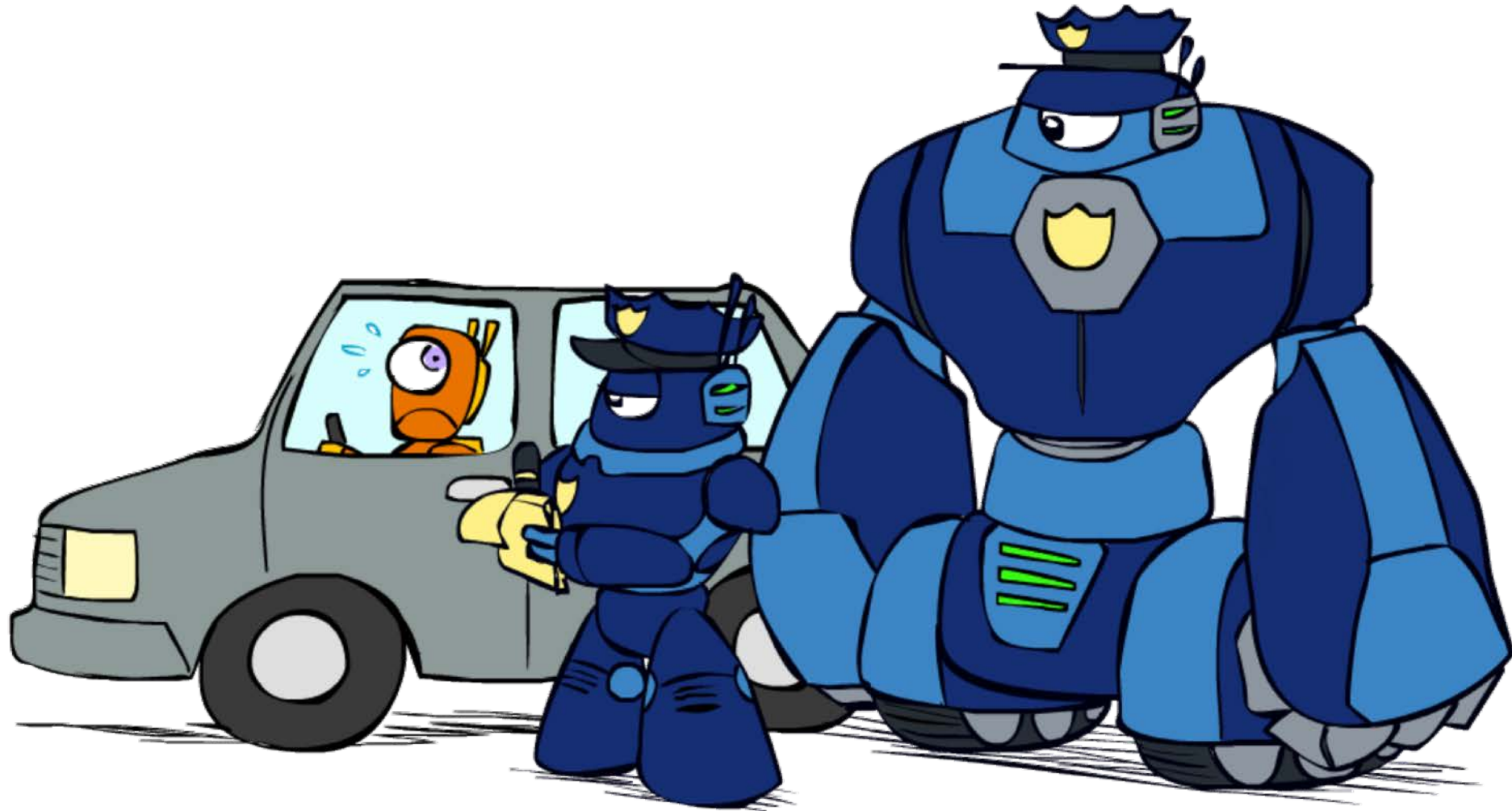


- Goals:

- Here: find any solution
- Also: find all, find best, etc.

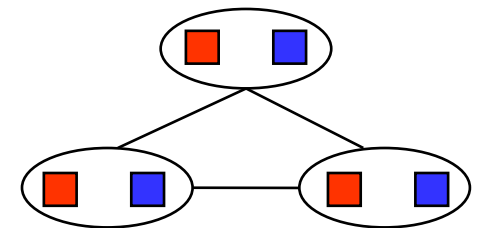
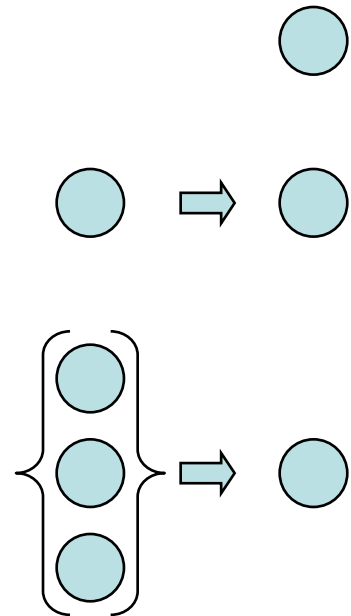


K-Consistency



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
- (You need to know the k=2 case: arc consistency)



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)

Ordering

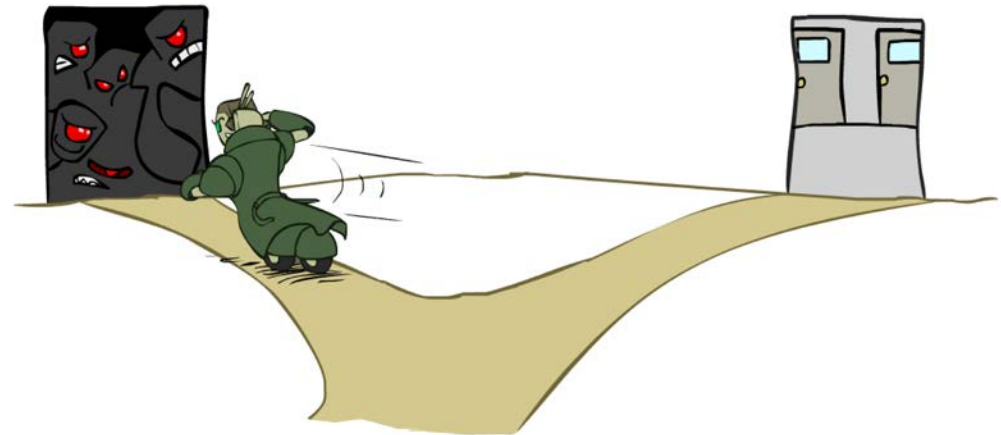


Ordering: Minimum Remaining Values

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

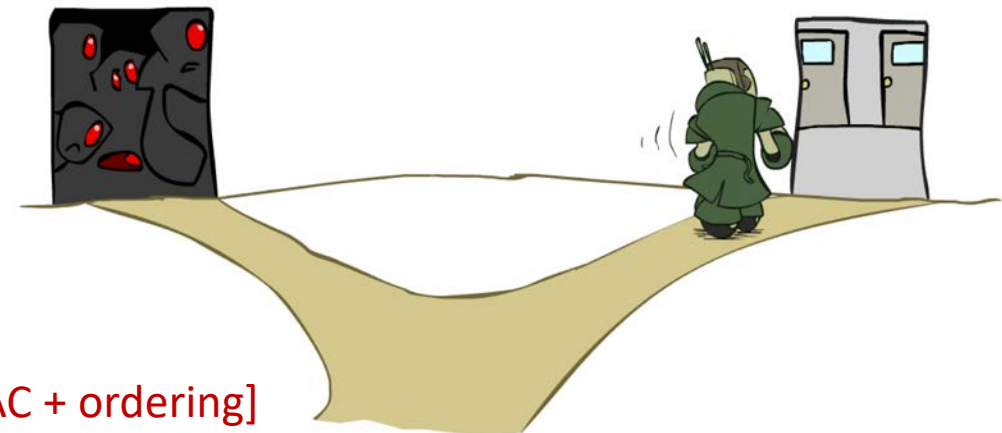
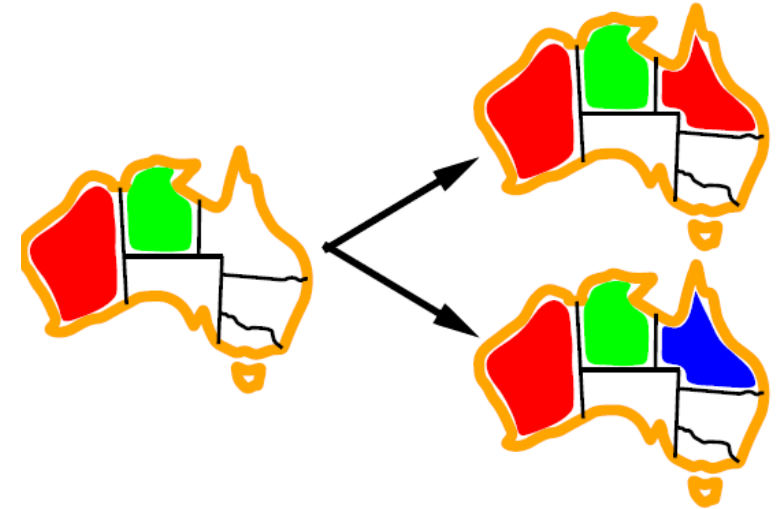


- Why min rather than max?
- Also called “most constrained variable”
- “Fail-fast” ordering



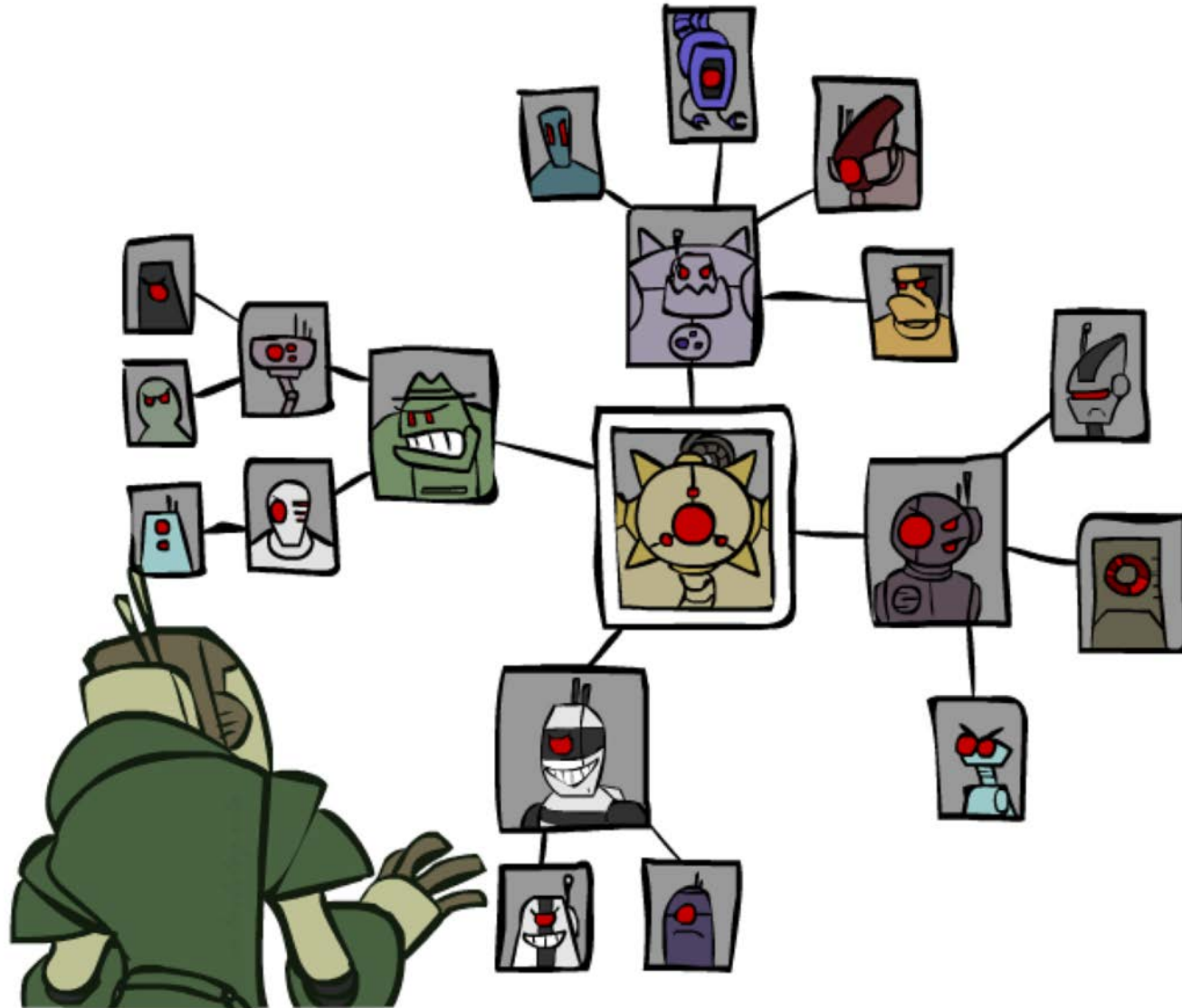
Ordering: Least Constraining Value

- 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)
- Why least rather than most?
- Combining these ordering ideas makes 1000 queens feasible



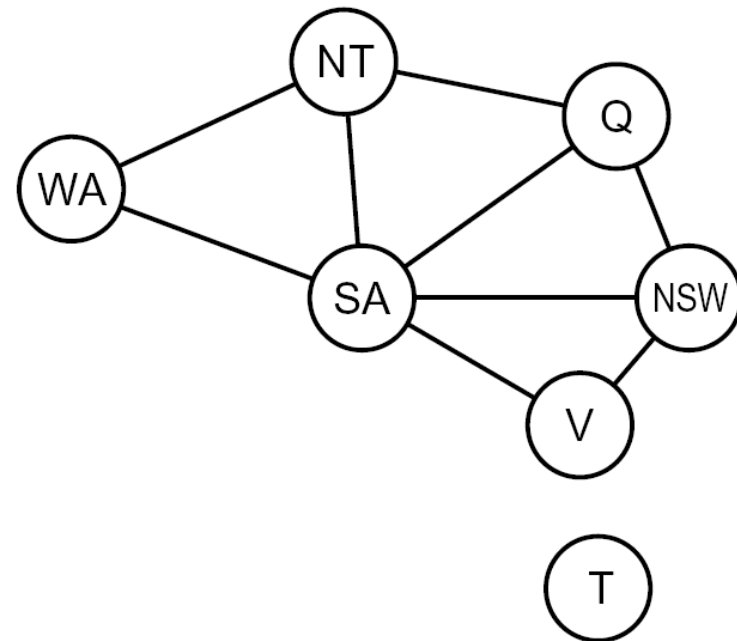
[Demo: coloring – backtracking + AC + ordering]

Structure

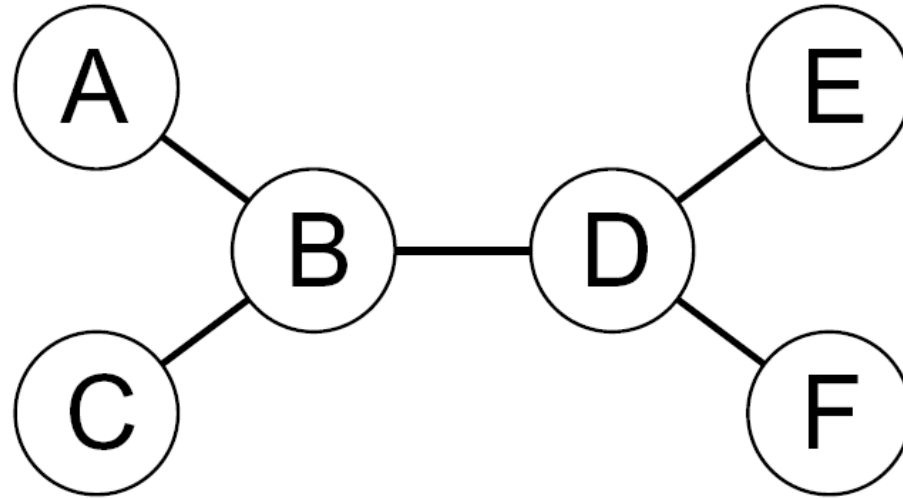


Problem Structure

- 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



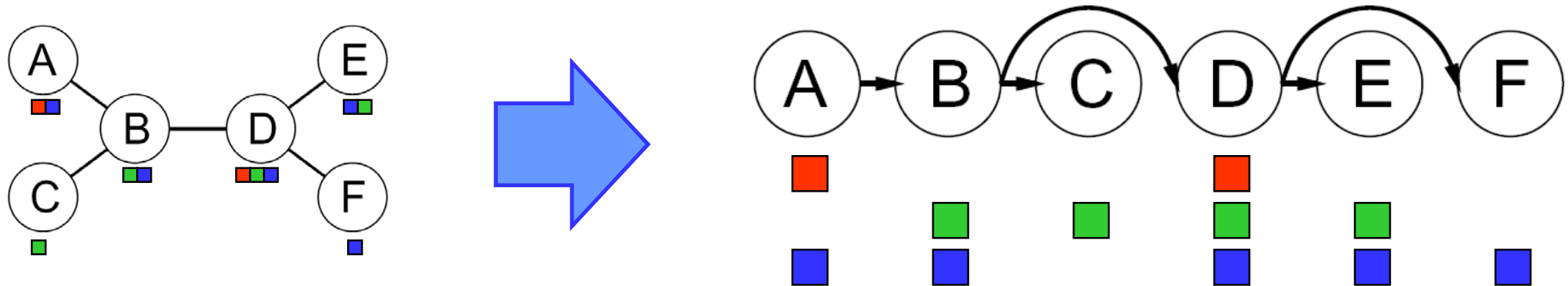
Tree-Structured CSPs



- 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

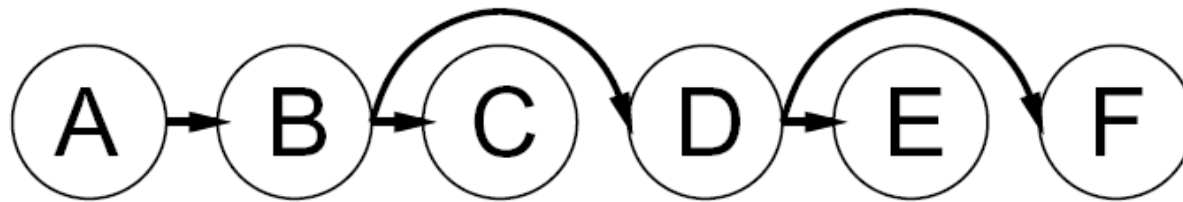


- Remove backward: For $i = n : 2$, apply $\text{RemoveInconsistent}(\text{Parent}(X_i), X_i)$
 - Assign forward: For $i = 1 : n$, assign X_i consistently with $\text{Parent}(X_i)$
- Runtime: $O(n d^2)$



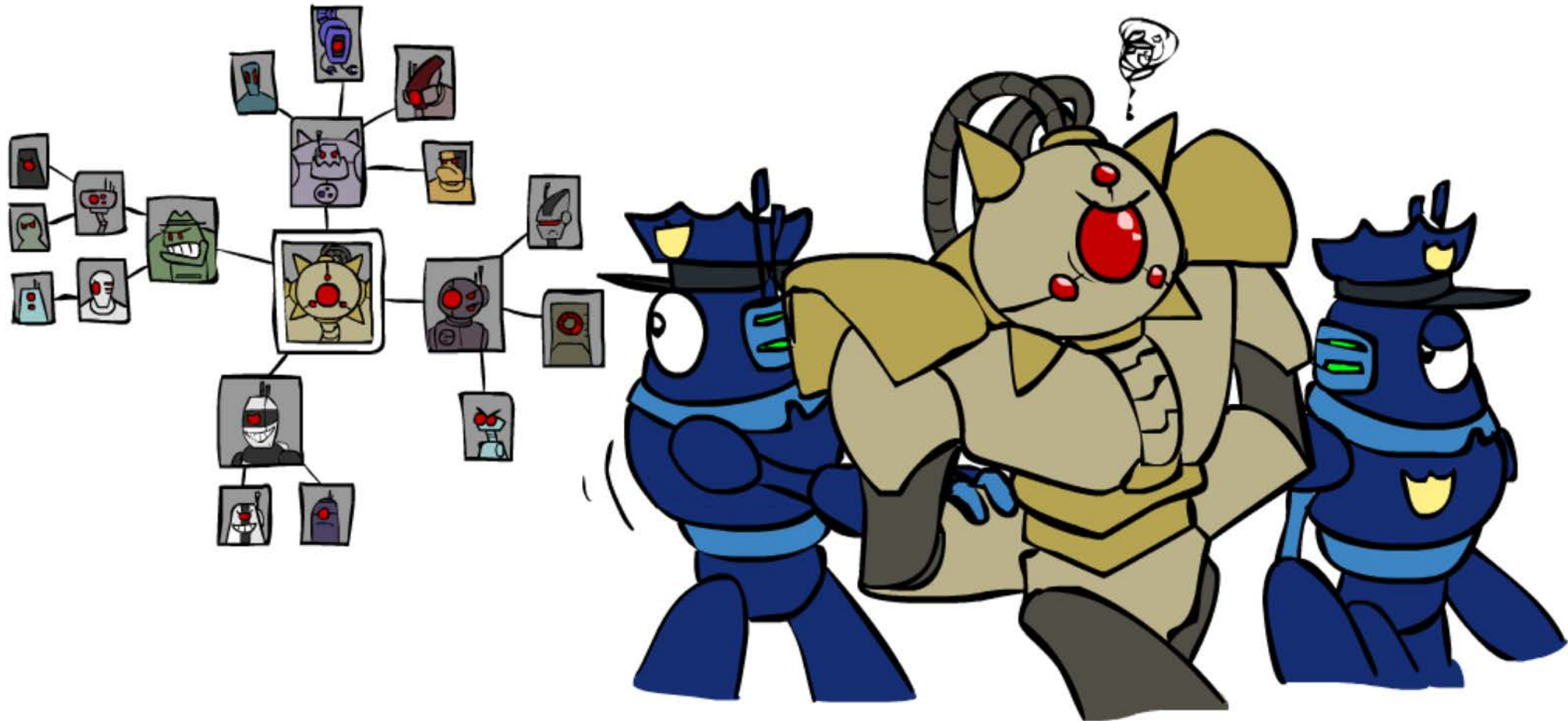
Tree-Structured CSPs

- 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)

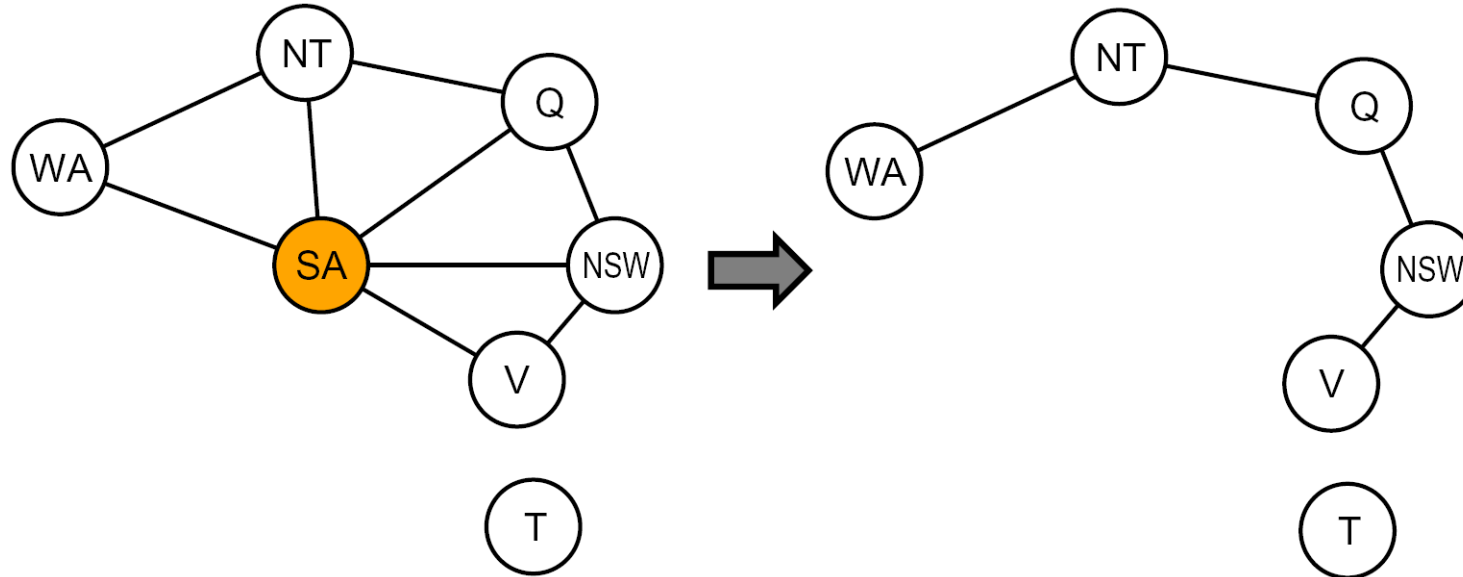


- Claim 2: If root-to-leaf arcs are consistent, forward assignment will not backtrack
- Proof: Induction on position
- Why doesn't this algorithm work with cycles in the constraint graph?
- Note: this basic idea is also used in Bayes' nets

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

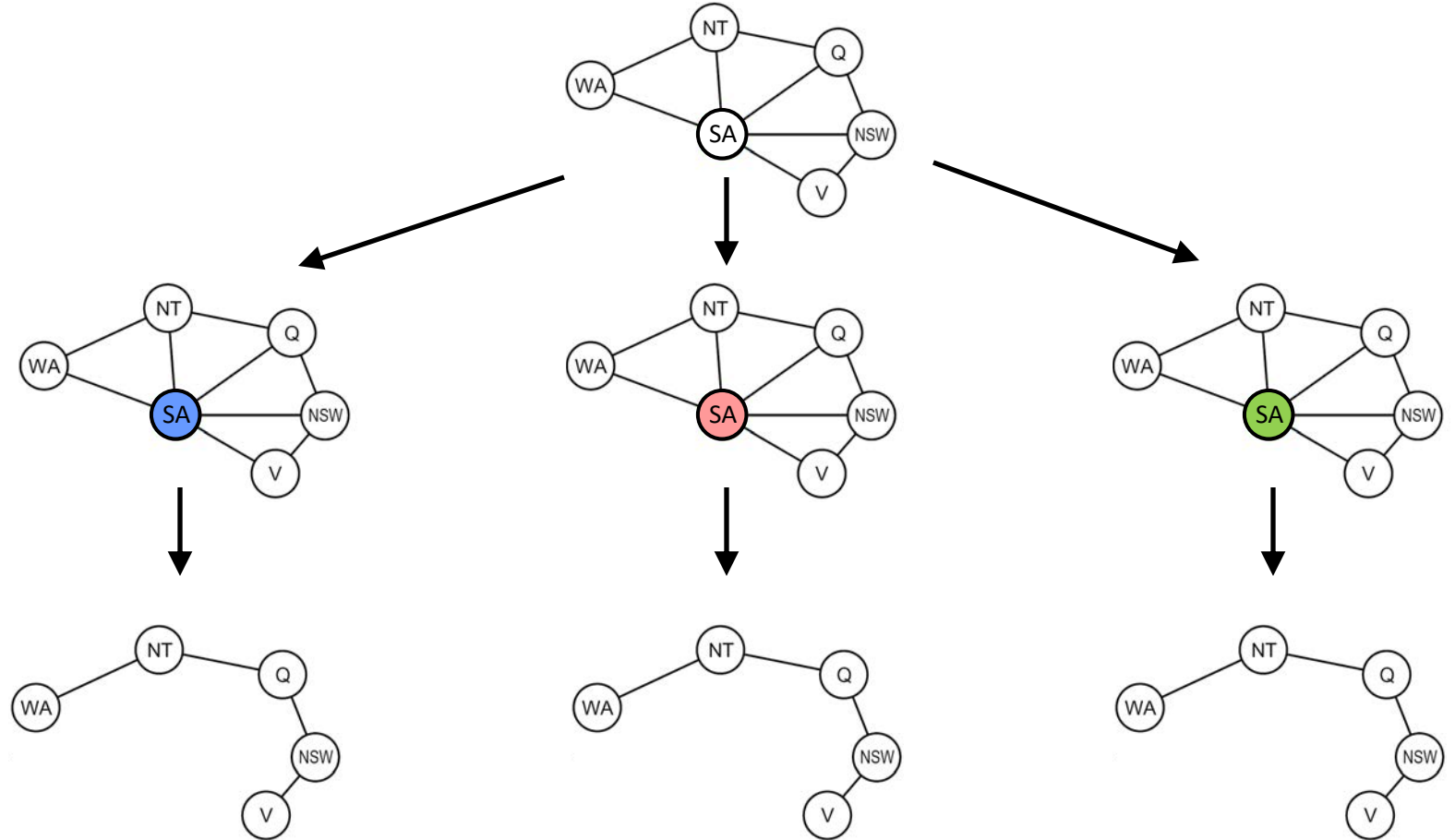
Cutset Conditioning

Choose a cutset

Instantiate the cutset
(all possible ways)

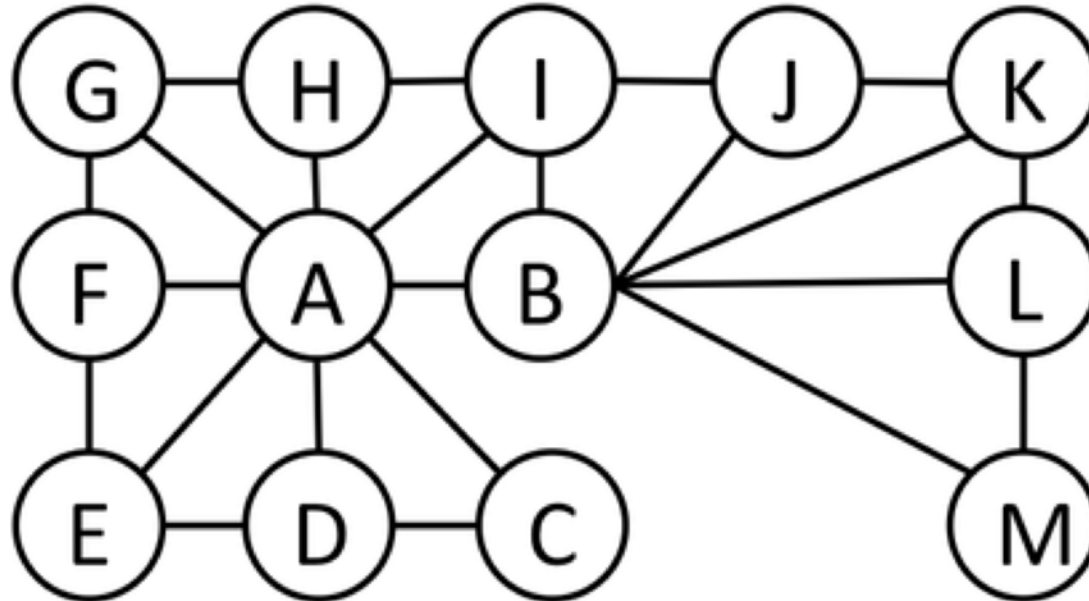
Compute residual CSP
for each assignment

Solve the residual CSPs
(tree structured)



Cutset Quiz

- Find the smallest cutset for the graph below.

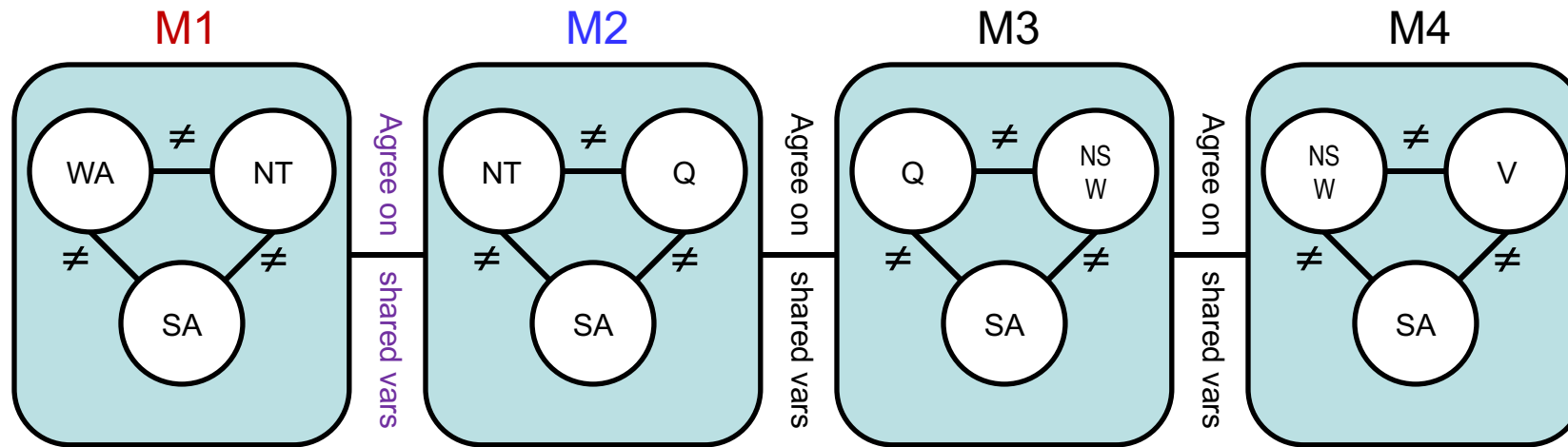
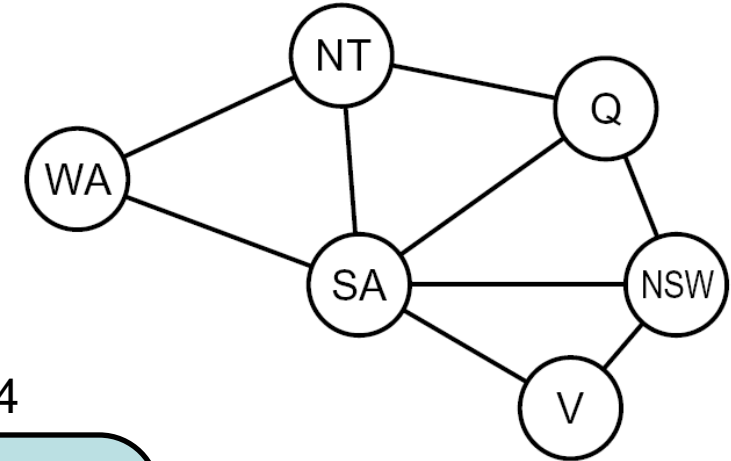


Exercise: Cutset

Consider a CSP with a constraint graph consisting of n variables arranged in a circle, where each variable has two constraints, one with each neighbor on either side. Explain how to solve this class of CSPs efficiently, in time $O(n)$.

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

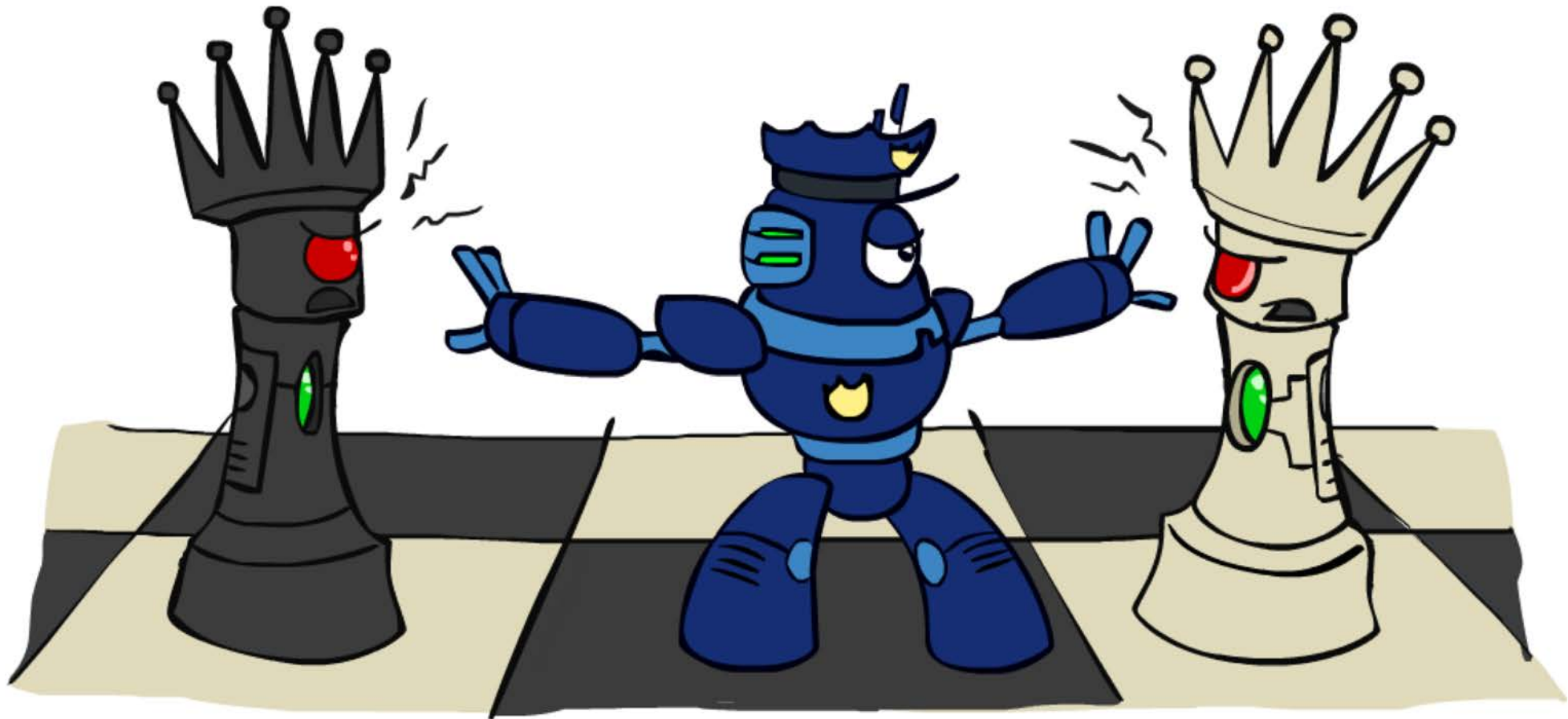


{(WA=r,SA=g,NT=b),
(WA=b,SA=r,NT=g),
...}

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

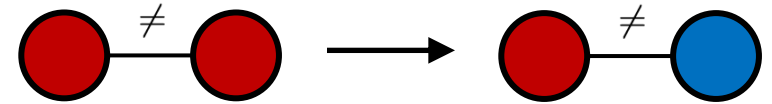
Agree: $(M1, M2) \in$
 $\{(WA=g,SA=g,NT=g), (NT=g,SA=g,Q=g), \dots\}$

Iterative Improvement

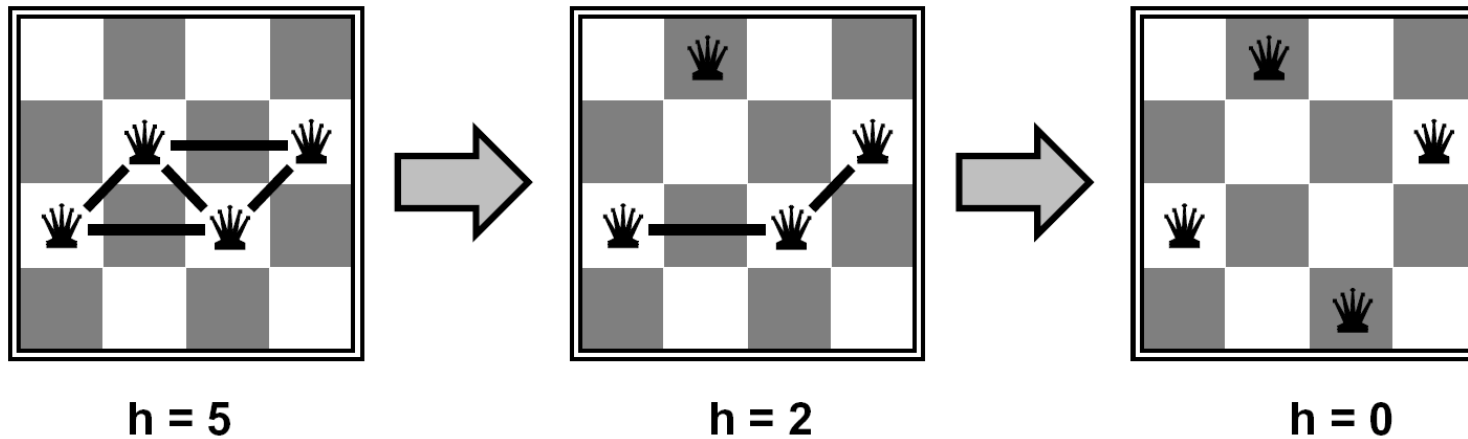


Iterative Algorithms (Local search) for CSPs

- 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



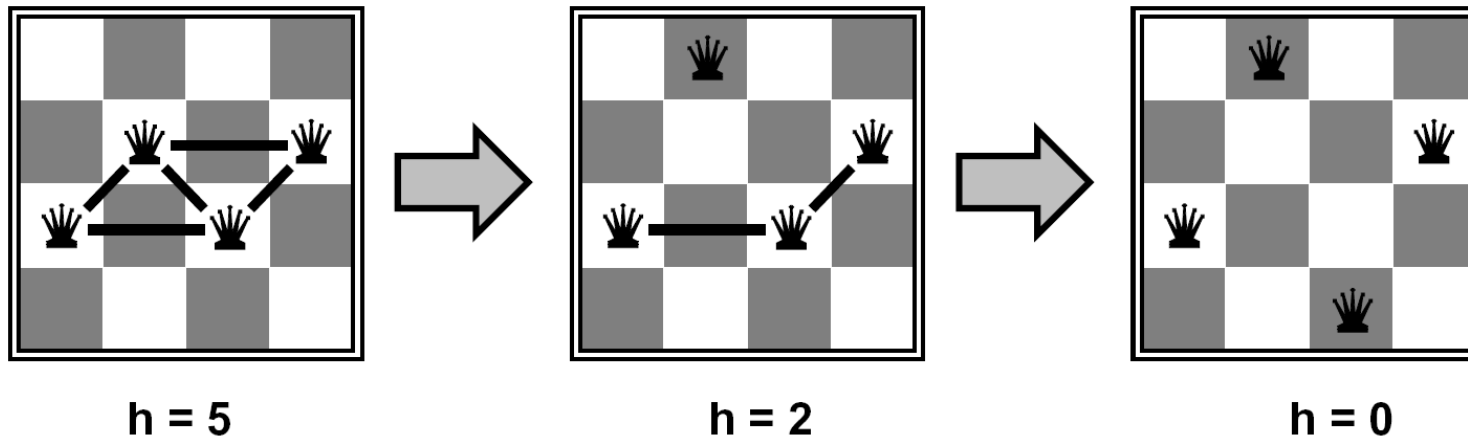
Example: 4-Queens



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

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

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

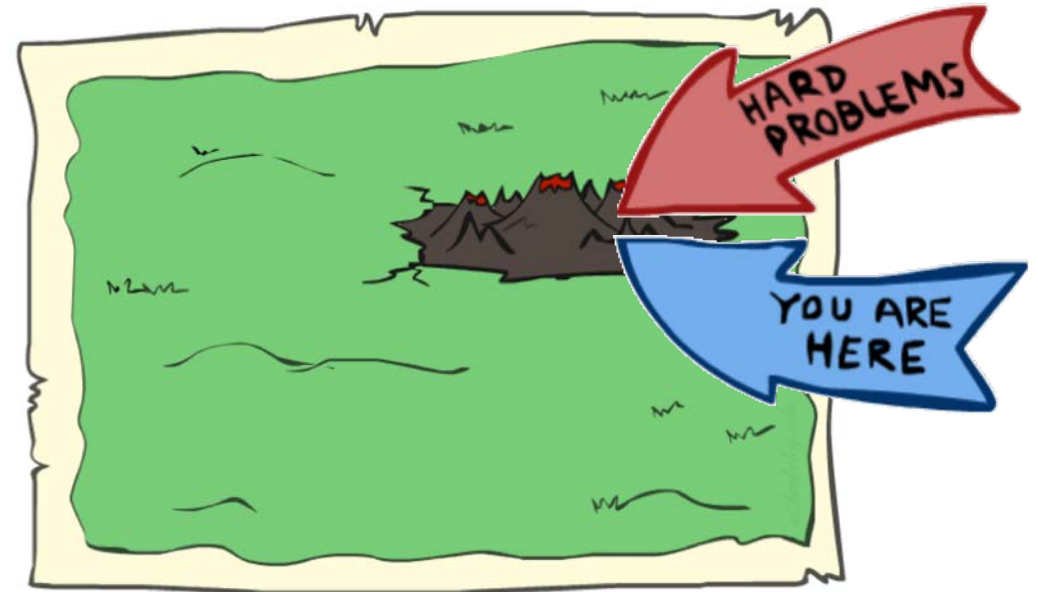
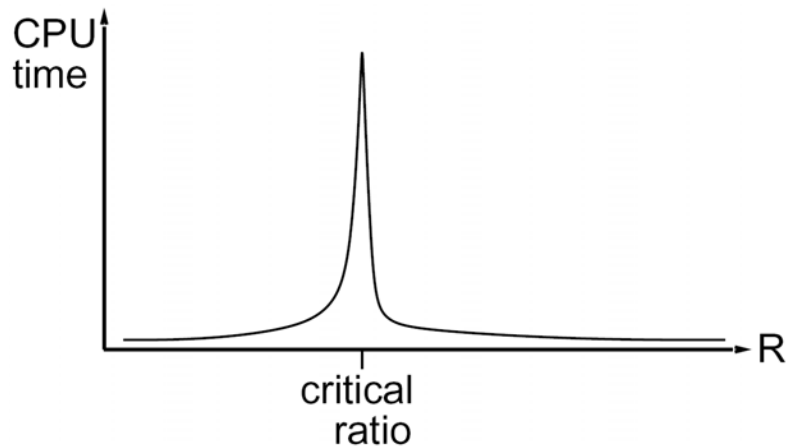
[Demo: n-queens – iterative improvement (L5D1)]
[Demo: coloring – iterative improvement]

How does Min-conflicts work in practice?

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

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



(your own illustration within Project 2)

Completeness of local search

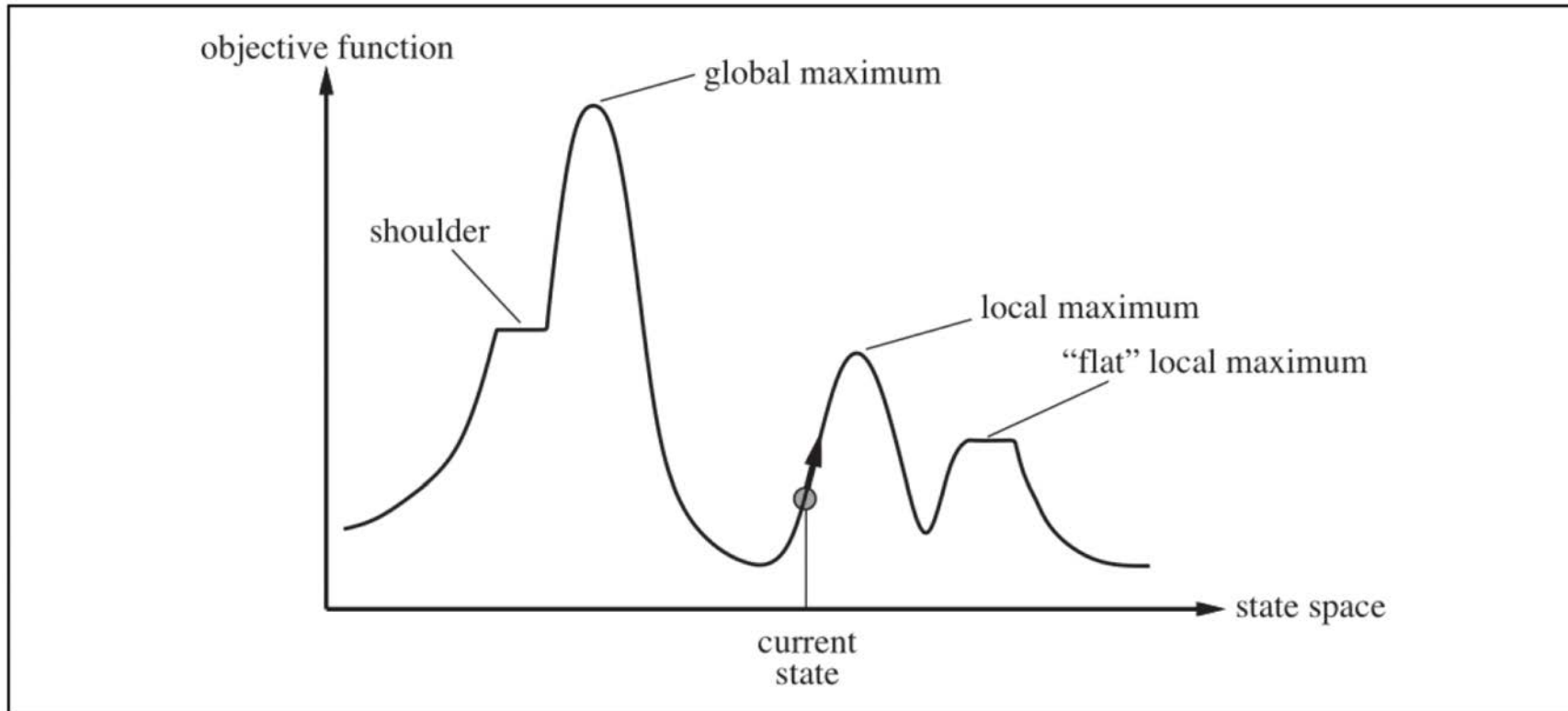
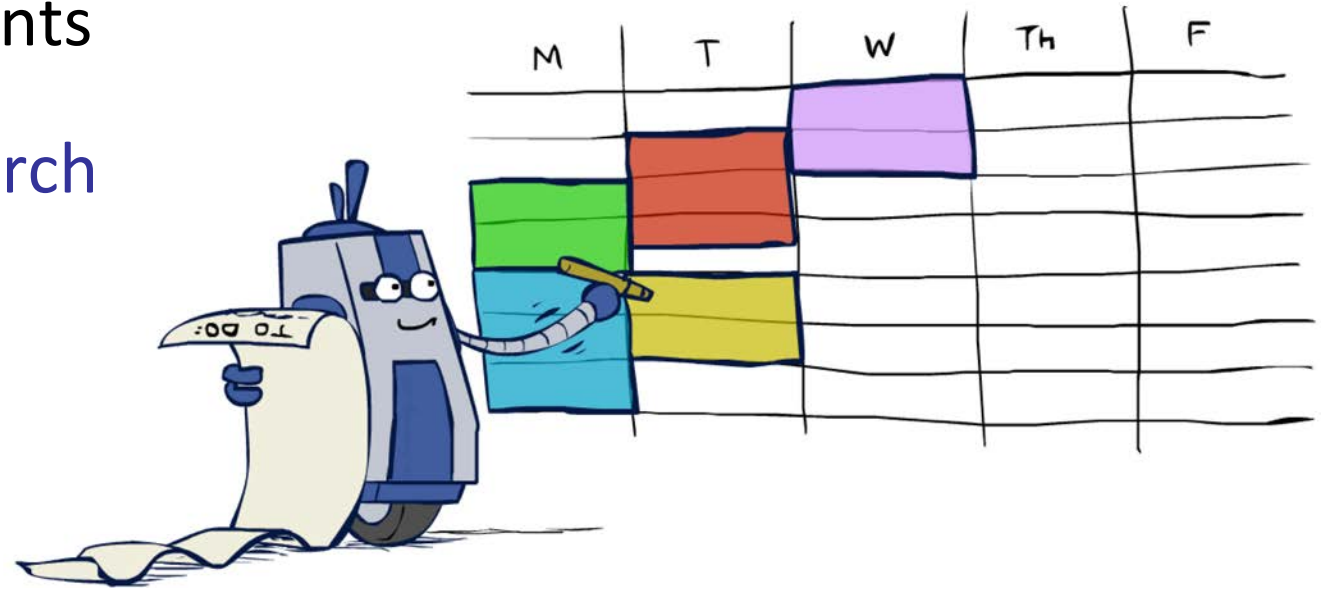


Figure 4.1 A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.

- Hill climbing with random restart, simulated annealing, genetic algorithms

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



HW: map coloring / performance

Generate random instances of map-coloring problems as follows: scatter n points on the unit square; select a point X at random, connect X by a straight line to the nearest point Y such that X is not already connected to Y and the line crosses no other line; repeat the previous step until no more connections are possible. The points represent regions on the map and the lines connect neighbors. Now try to find k -colorings of each map, for both $k=3$ and $k=4$, using min-conflicts, backtracking, backtracking with forward checking, and backtracking with MAC. Construct a table of average run times for each algorithm for values of n up to the largest you can manage. Comment on your results.

HW: Critical ratio

Using a CSP solver program and another program to generate random problem instances of CSPs, report on the time to solve the problem as a function of the ratio of the number of constraints to the number of variables.

Next Time: Adversarial Search!
