

Announcement

- TA office hours @SIST 1B-105

- | | | |
|-------------|-------------|-----|
| ■ Tuesday | 20:00-21:00 | 陈天素 |
| ■ Wednesday | 20:00-21:00 | 陈卓 |
| ■ Friday | 15:00-16:00 | 吴昊一 |
| ■ Friday | 16:00-17:00 | 杨松霖 |

Constraint Satisfaction Problems

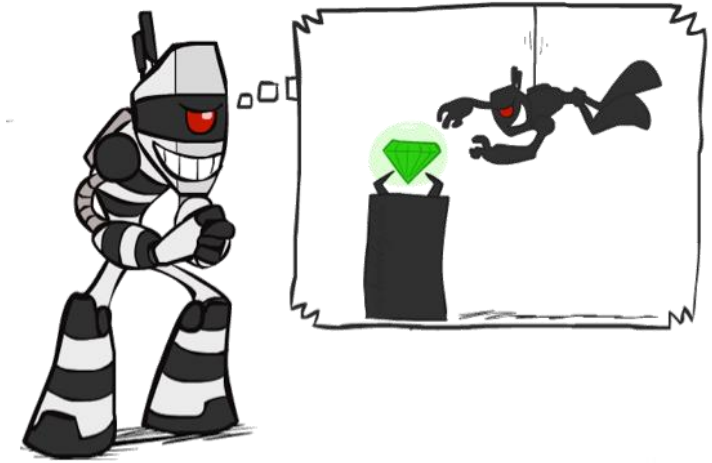


AIMA Chapter 6

What is Search For?

- Planning: (optimal) sequences of actions

- The path to the goal is the important thing
- Paths have various costs, depths
- Heuristics give problem-specific guidance



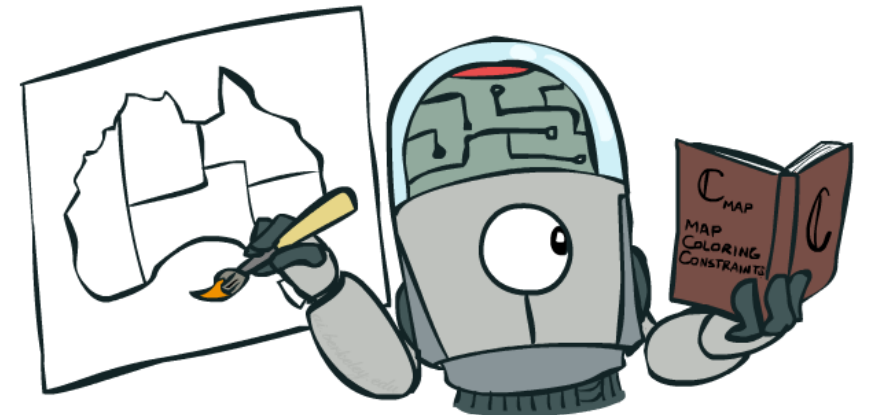
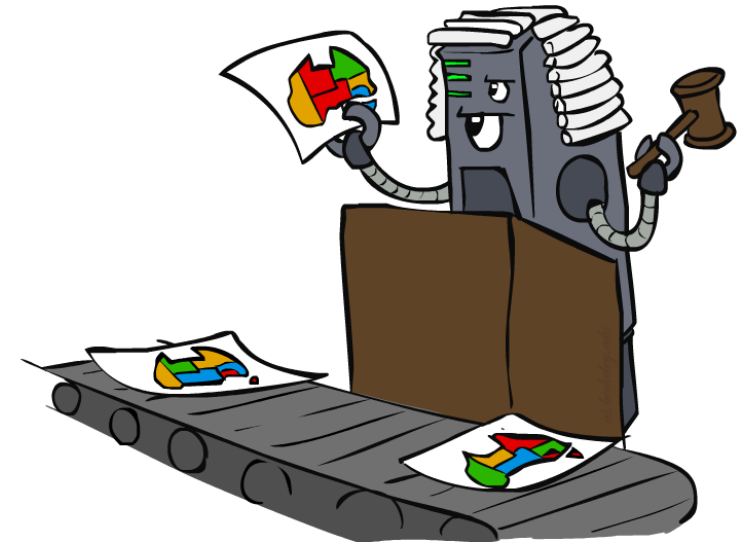
- Identification

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

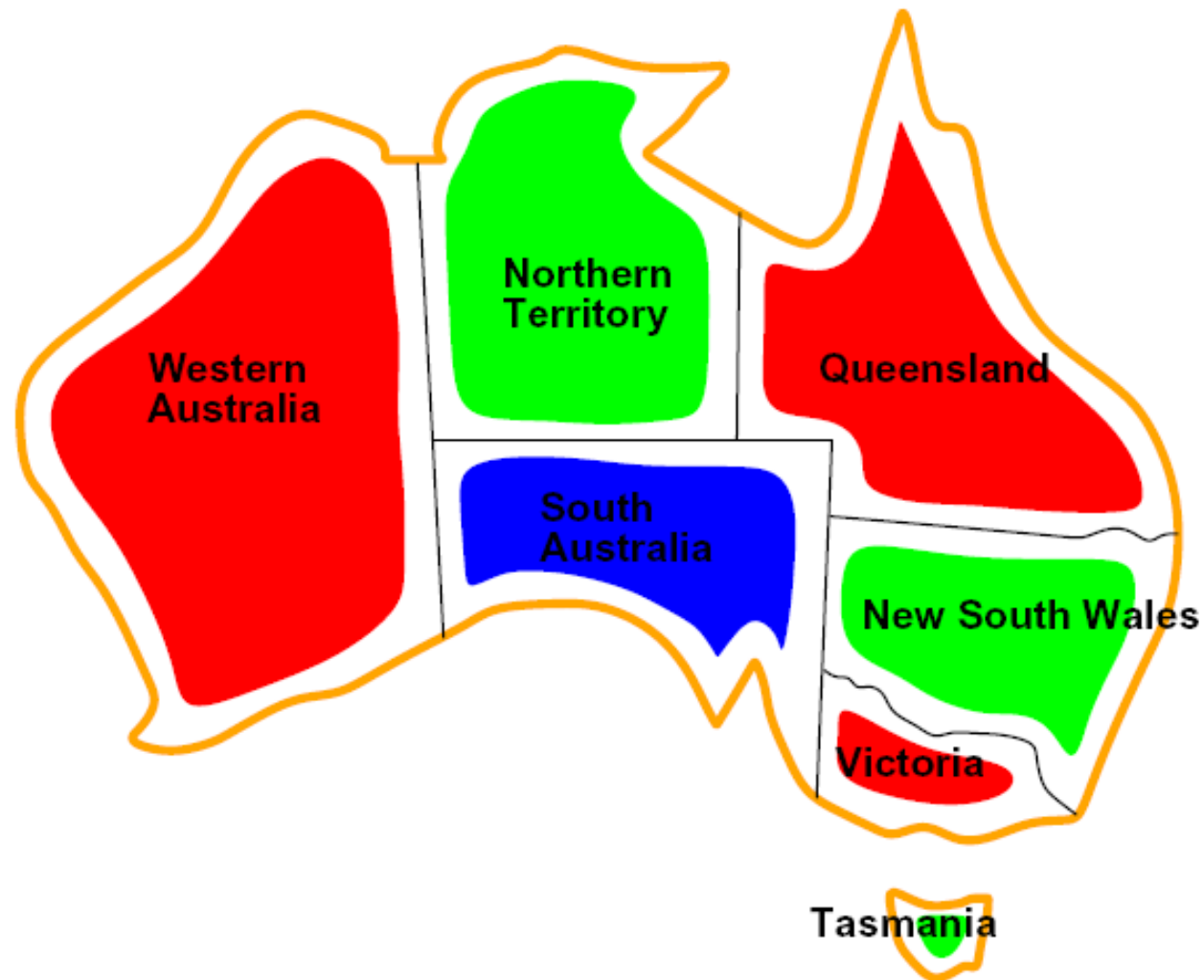


Search Problems

- Planning = 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
- Identification = 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



CSP Examples



Example: Map Coloring

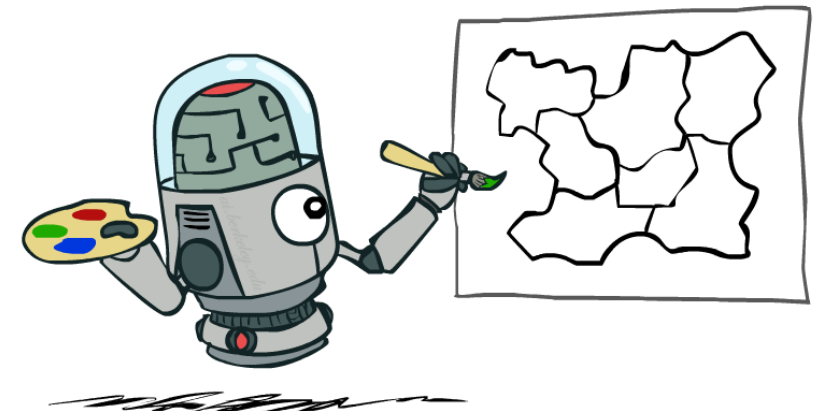
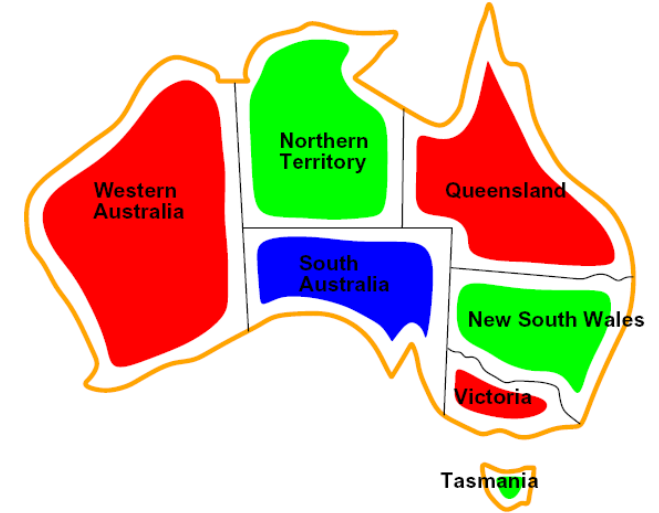
- Variables: WA, NT, Q, NSW, V, SA, T
- Domains: $D = \{\text{red, green, blue}\}$
- Constraints: adjacent regions must have different colors

Implicit: $WA \neq NT$

Explicit: $(WA, NT) \in \{(\text{red, green}), (\text{red, blue}), \dots\}$

- Solutions are assignments satisfying all constraints, e.g.:

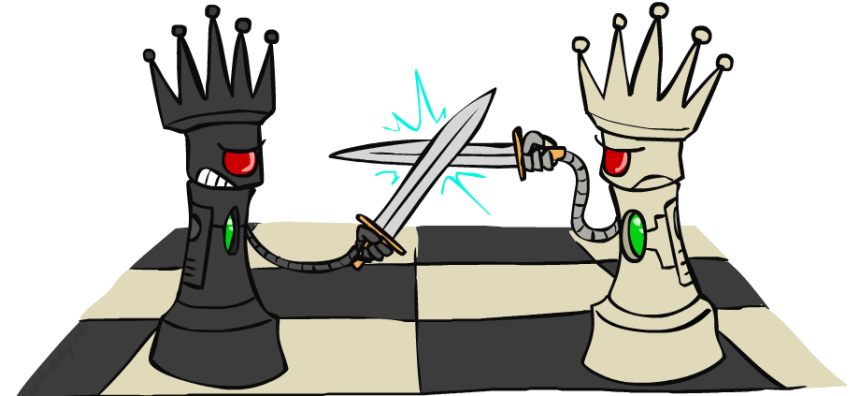
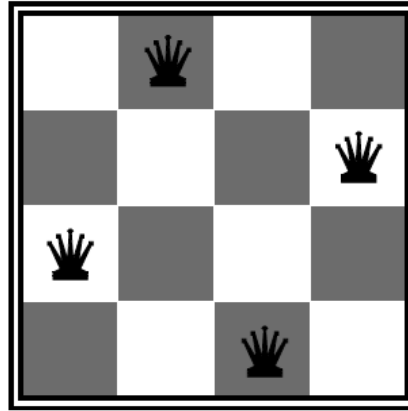
$\{WA=\text{red}, NT=\text{green}, Q=\text{red}, NSW=\text{green}, V=\text{red}, SA=\text{blue}, T=\text{green}\}$



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:

- Variables: Q_k

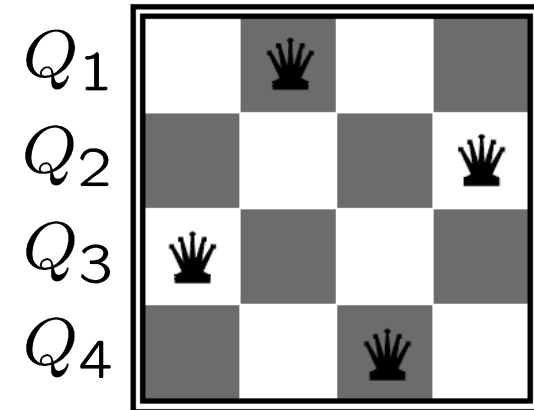
- Domains: $\{1, 2, 3, \dots, N\}$

- Constraints:

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

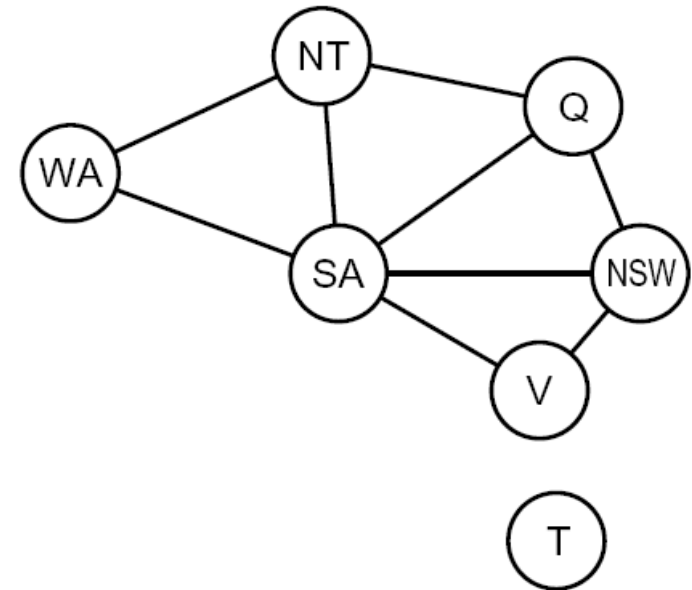
Explicit: $(Q_1, Q_2) \in \{(1, 3), (1, 4), \dots\}$

...



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?



Example: Cryptarithmic

- Variables:

$F T U W R O X_1 X_2 X_3$

- Domains:

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

- Constraints:

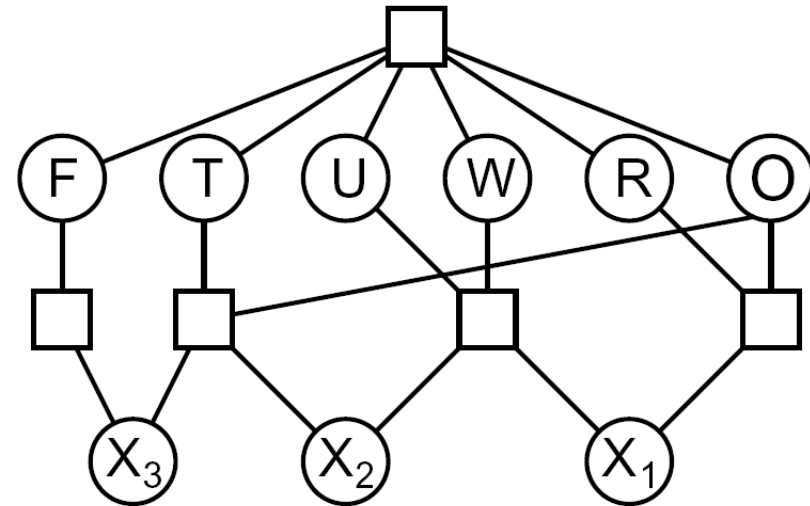
$\text{alldiff}(F, T, U, W, R, O)$

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

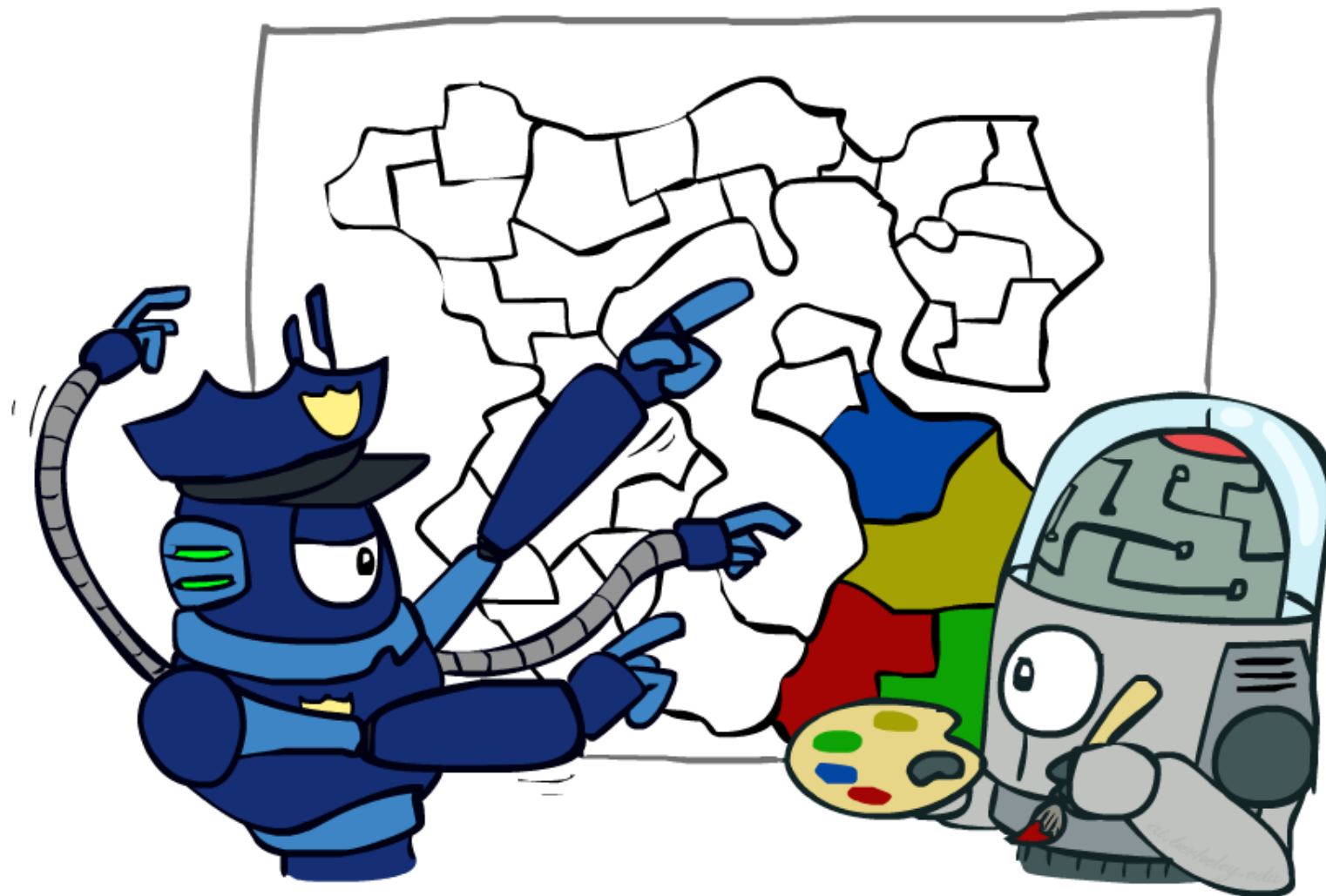
\dots

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

X_1



Varieties of CSPs and Constraints



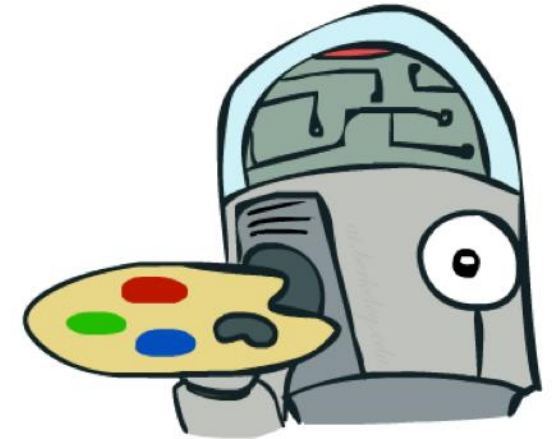
Varieties of CSPs

- 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.:

$$SA \neq \text{green}$$

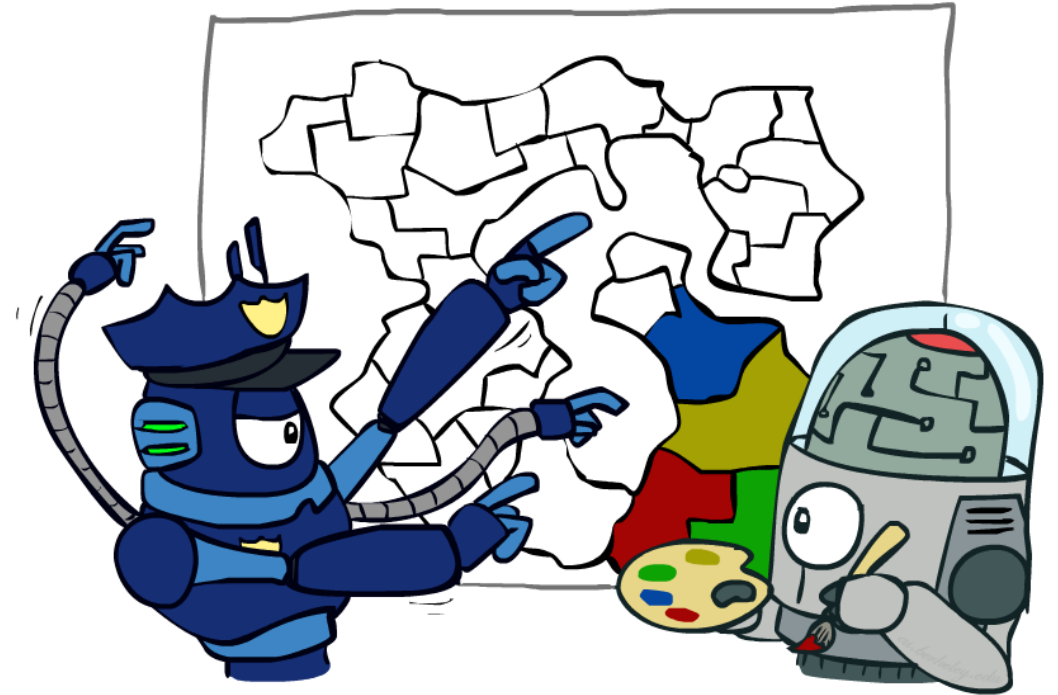
- Binary constraints involve pairs of variables, e.g.:

$$SA \neq WA$$

- Higher-order constraints involve 3 or more variables:

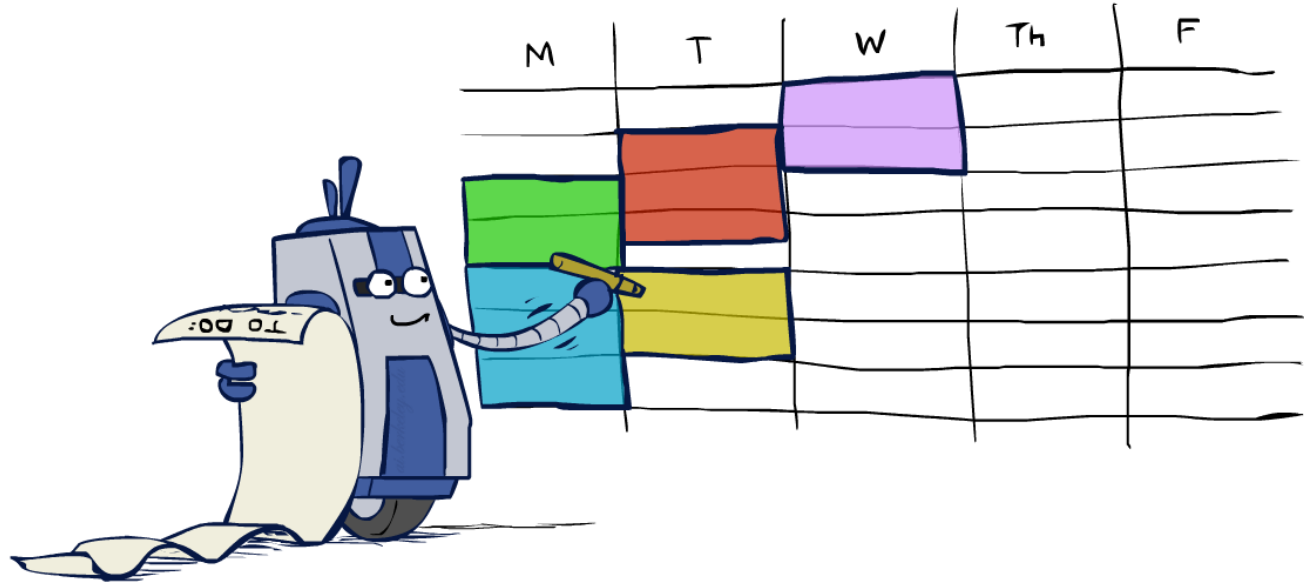
- Preferences (soft constraints):

- 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!



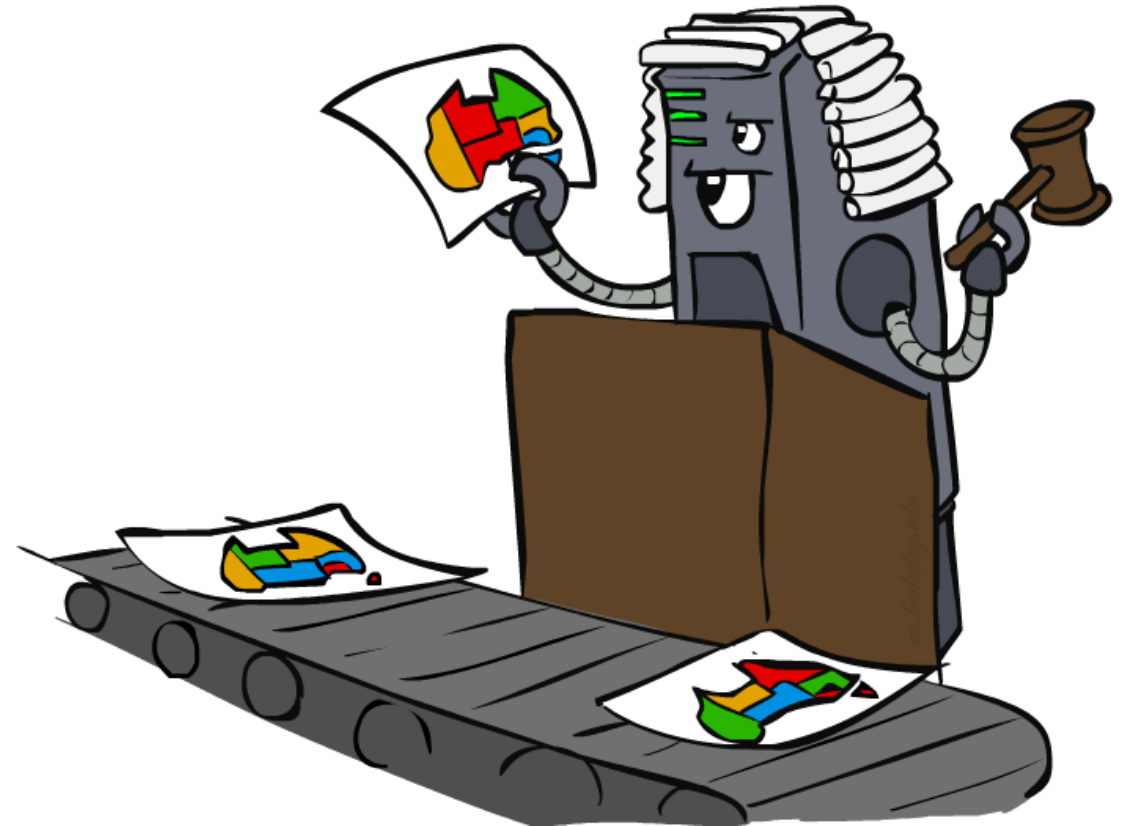
- Many real-world problems involve real-valued variables...

Solving CSPs



Standard Search Formulation

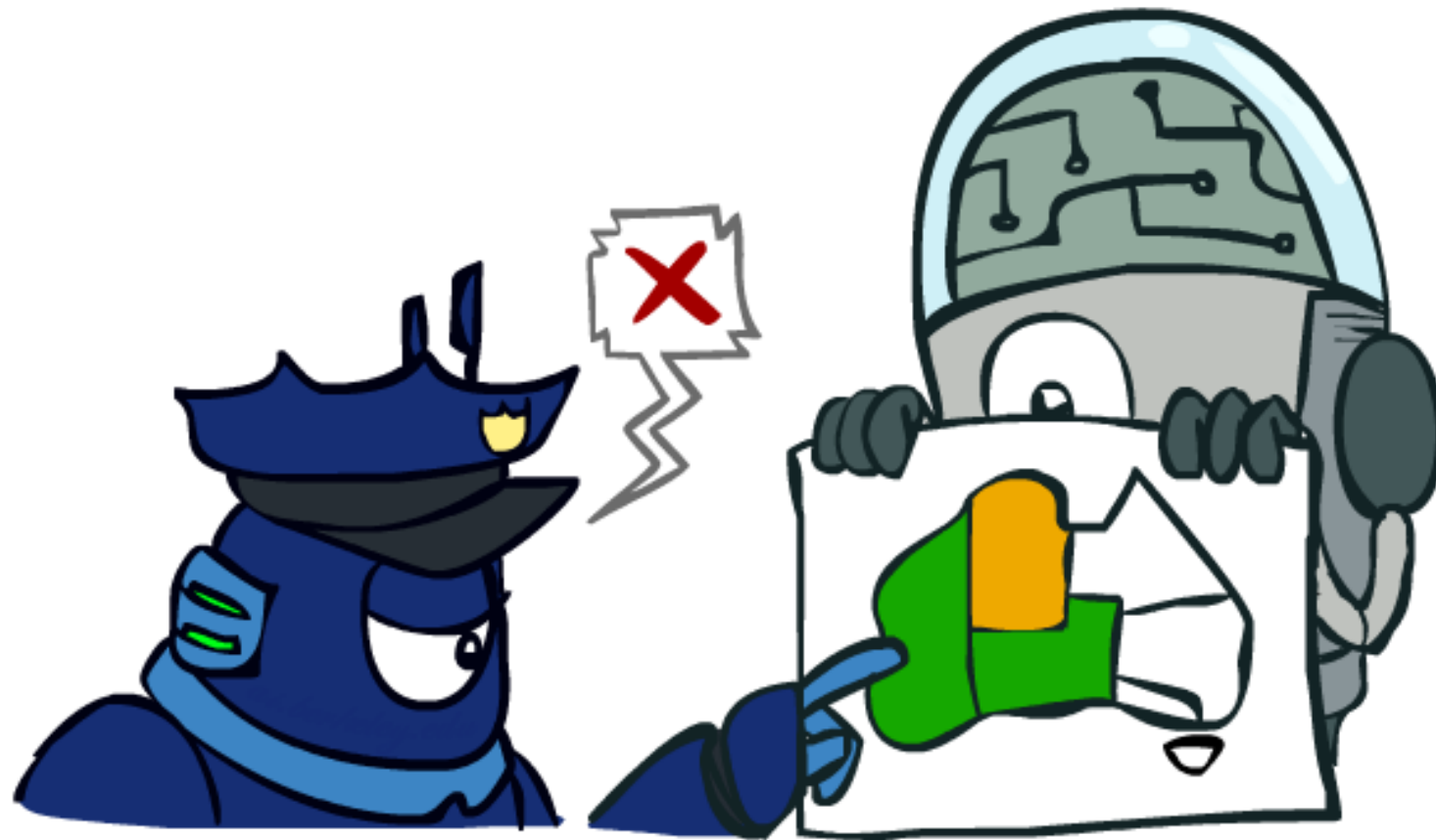
- Standard search formulation of CSPs
- States defined by the values assigned so far (partial assignments)
 - Initial state: the empty assignment, $\{\}$
 - Successor function: assign a value to an unassigned variable
 - Goal test: the current assignment is complete and satisfies all constraints



Search Methods

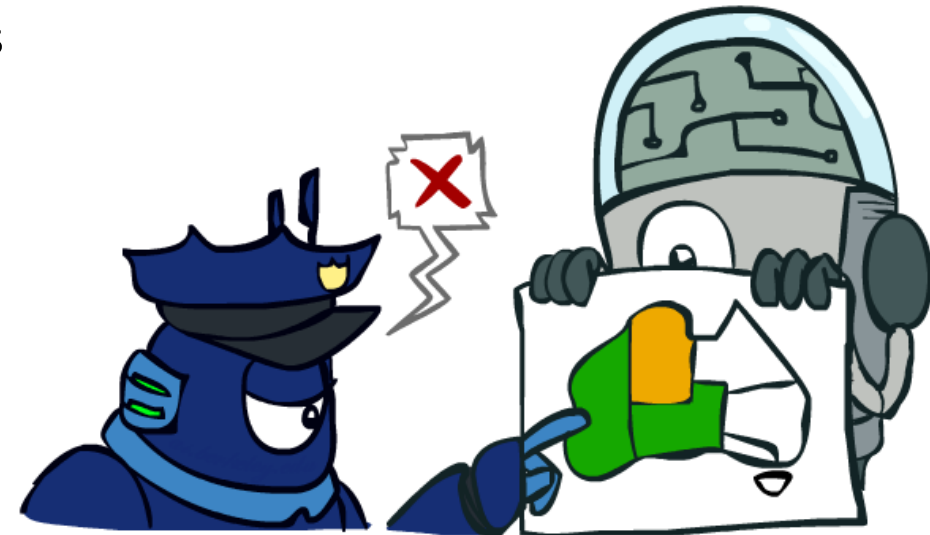
- What would DFS do?
 - Demo
 - What's wrong?

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 $n \approx 25$



Demo – Backtracking

Backtracking Search

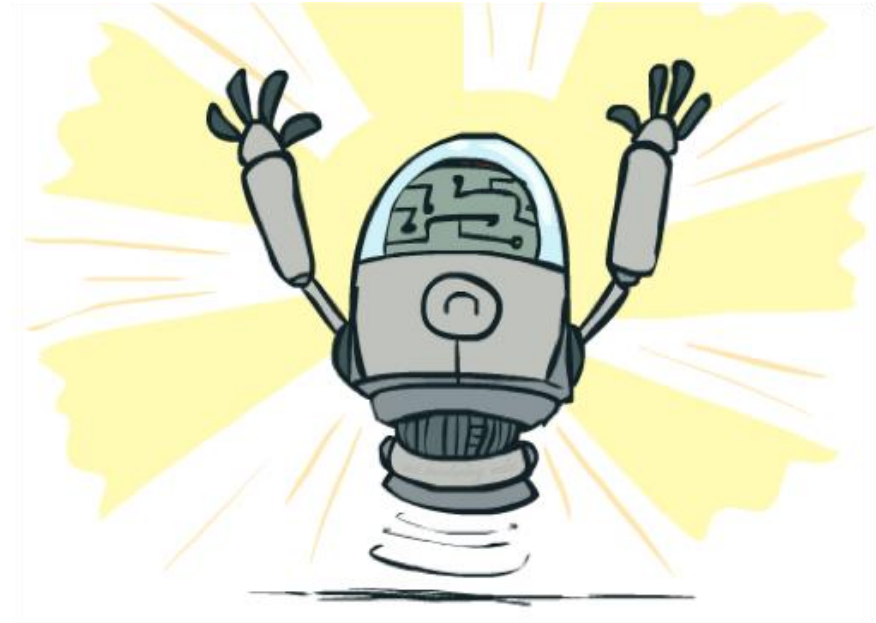
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

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

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?

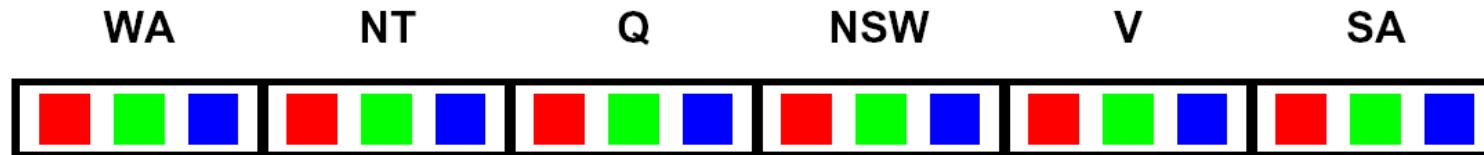


Filtering



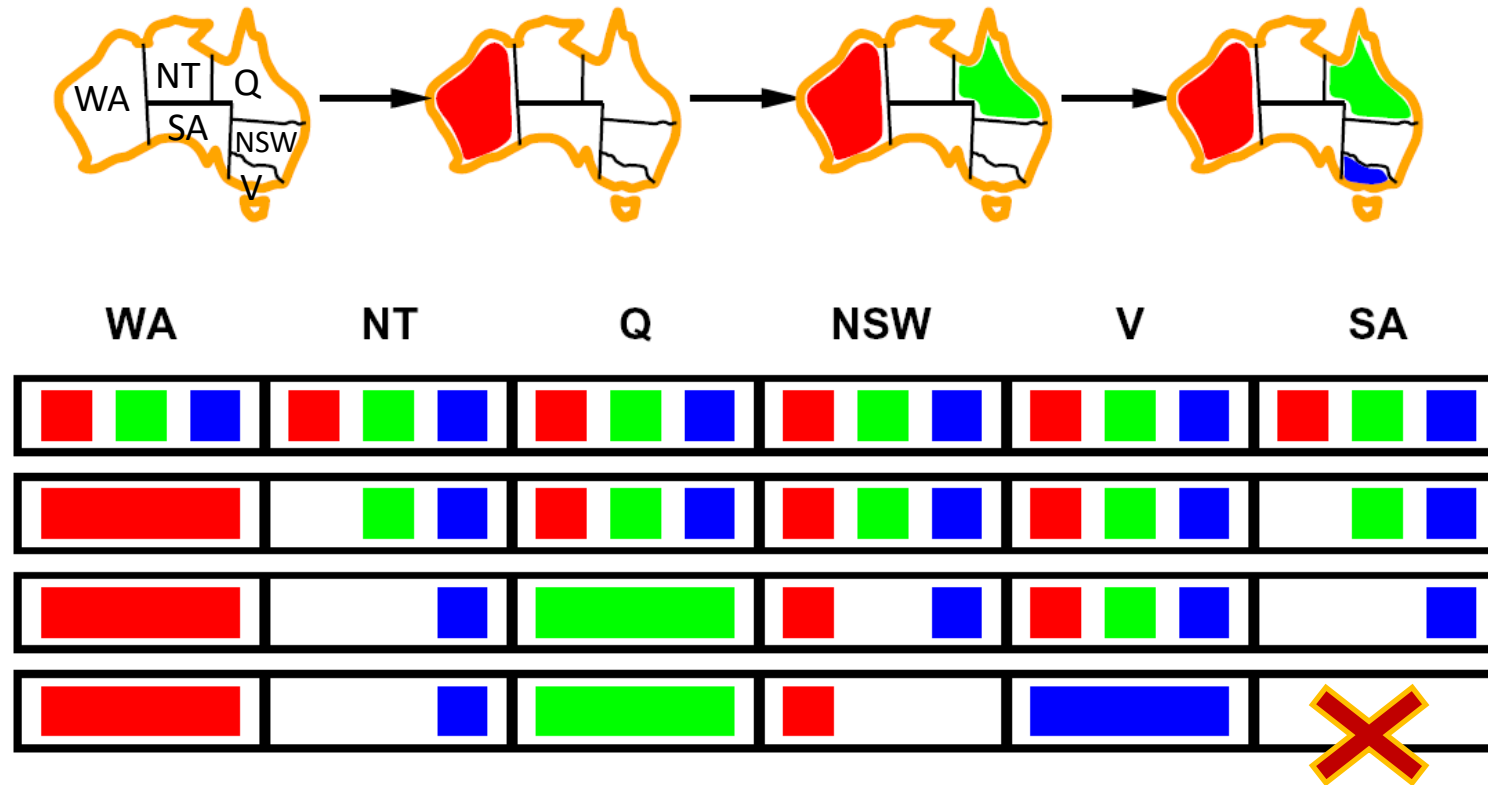
Filtering: Forward Checking

- 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



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

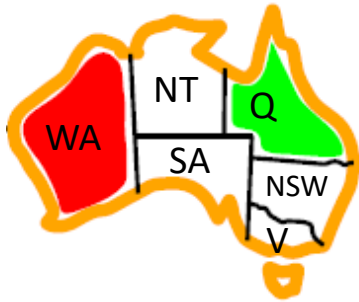


Demo

- 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:

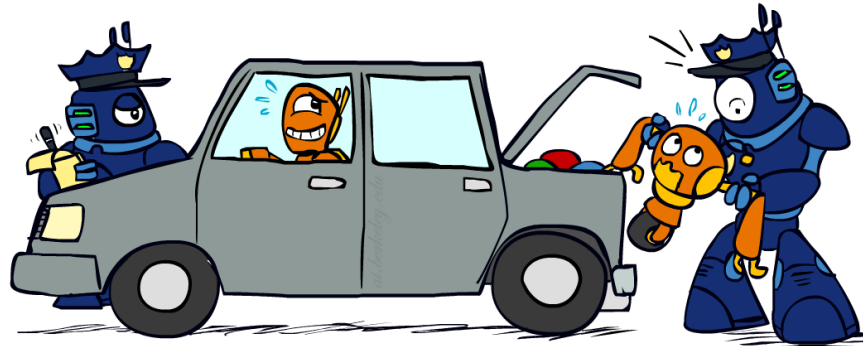
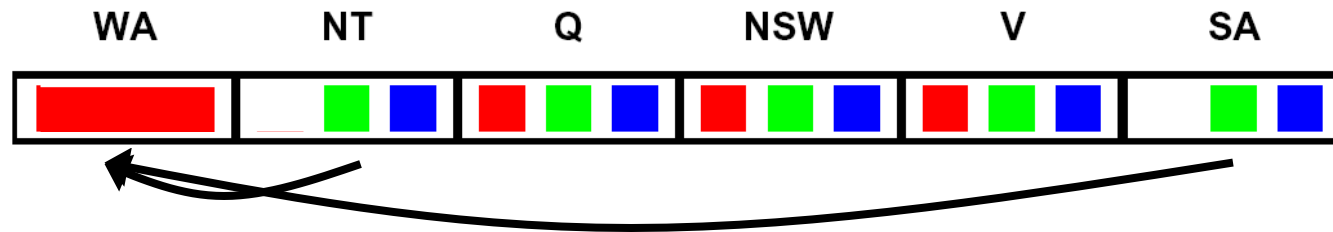
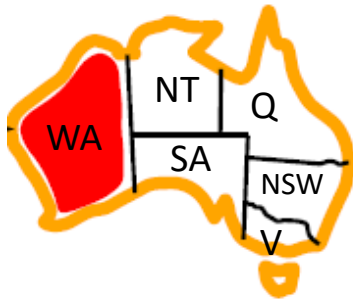


WA	NT	Q	NSW	V	SA
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

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

Consistency of A Single Arc

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

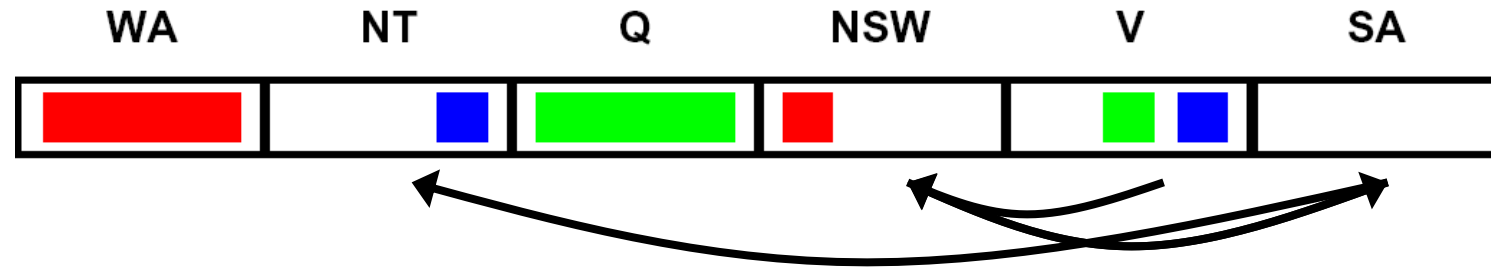
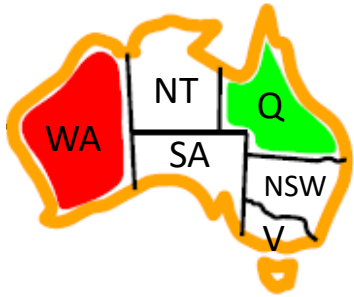


Forward checking?

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 $X \rightarrow Y$ 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!*

Enforcing Arc Consistency in a CSP

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

while queue is not empty do
     $(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\textit{queue})$ 
    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



---

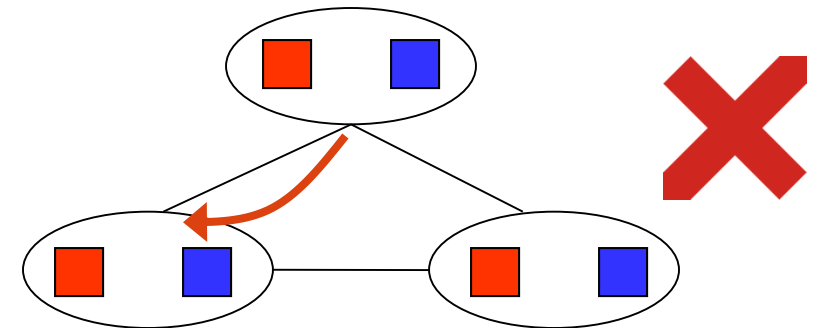
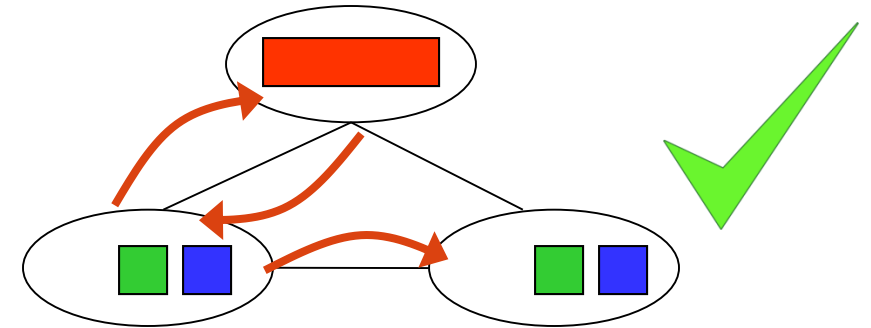


function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) returns true iff succeeds
    removed  $\leftarrow$  false
    for each  $x$  in DOMAIN[ $X_i$ ] do
        if no value  $y$  in DOMAIN[ $X_j$ ] allows  $(x, y)$  to satisfy the constraint  $X_i \leftrightarrow X_j$ 
            then delete  $x$  from DOMAIN[ $X_i$ ]; removed  $\leftarrow$  true
    return removed
```

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

Limitations of Arc Consistency

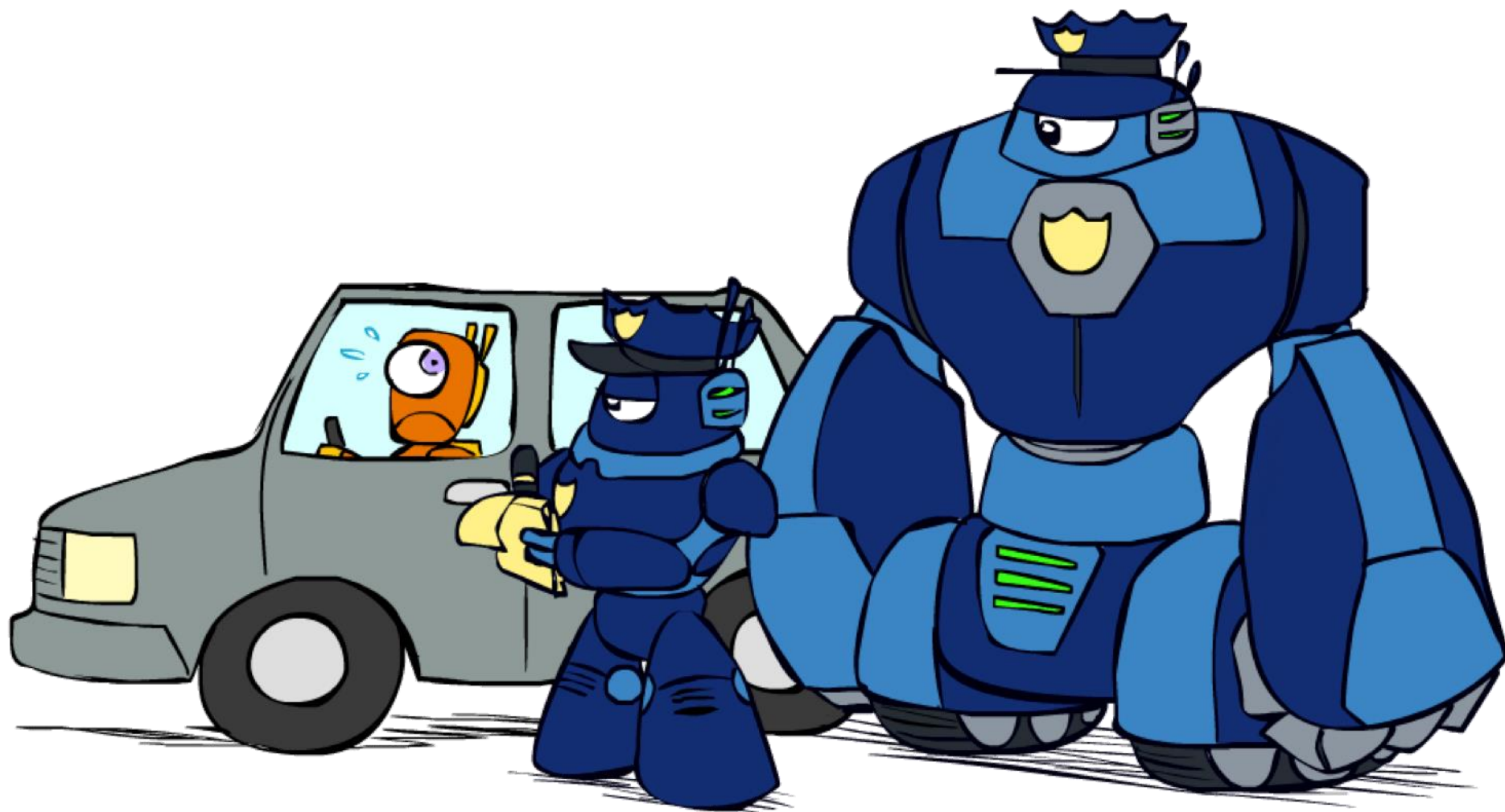
- After enforcing arc consistency:
 - Can have one solution left
 - Can have multiple solutions left
 - Can have no solutions left (and not know it)
- Arc consistency still runs inside a backtracking search!



Demo

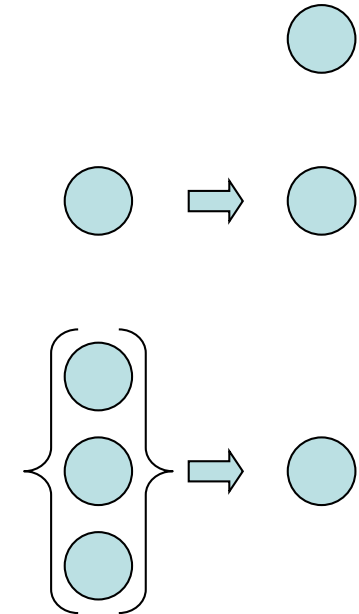
- Backtracking with Forward Checking
- Backtracking with Arc Consistency

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 kth node.
- Higher k more expensive to compute



Strong K-Consistency

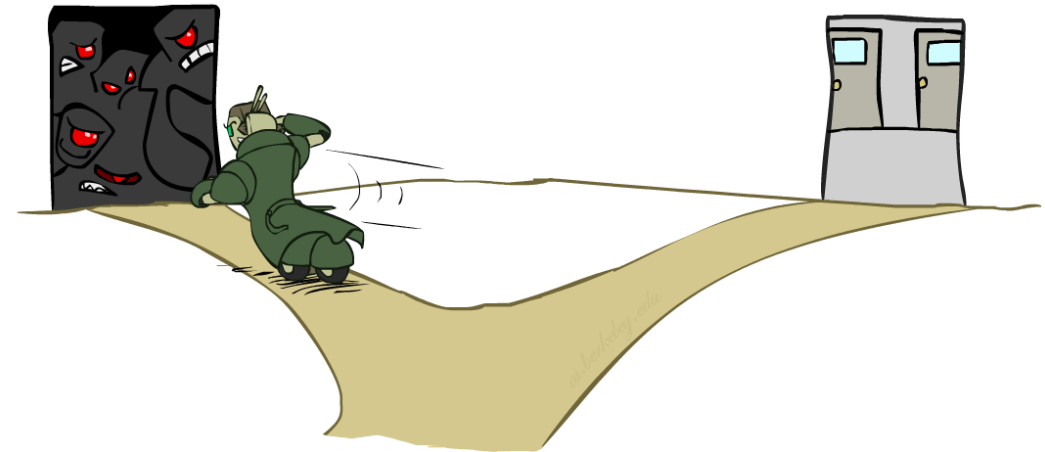
- Strong k-consistency: also k-1, k-2, ... 1 consistent
- Claim: strong n-consistency means we can solve without backtracking!
- Why?
 - Choose any assignment to any variable
 - Choose a new variable
 - By 2-consistency, there is a choice consistent with the first
 - Choose a new variable
 - By 3-consistency, 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)

Ordering



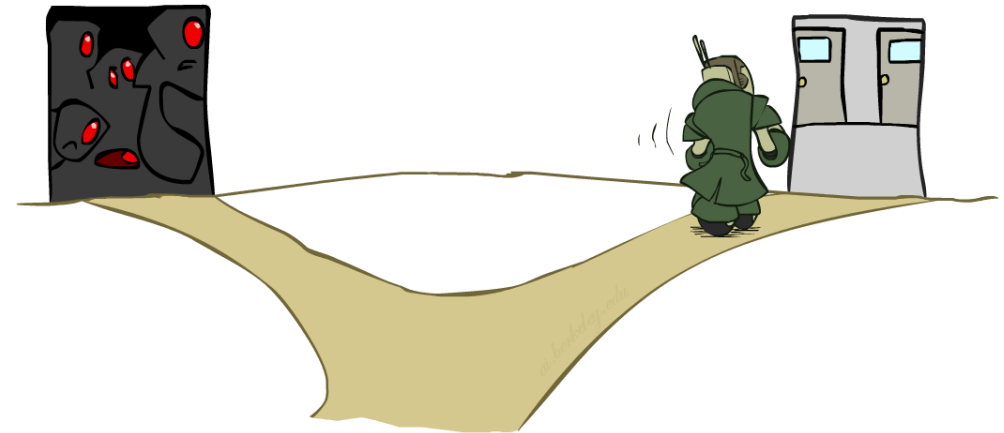
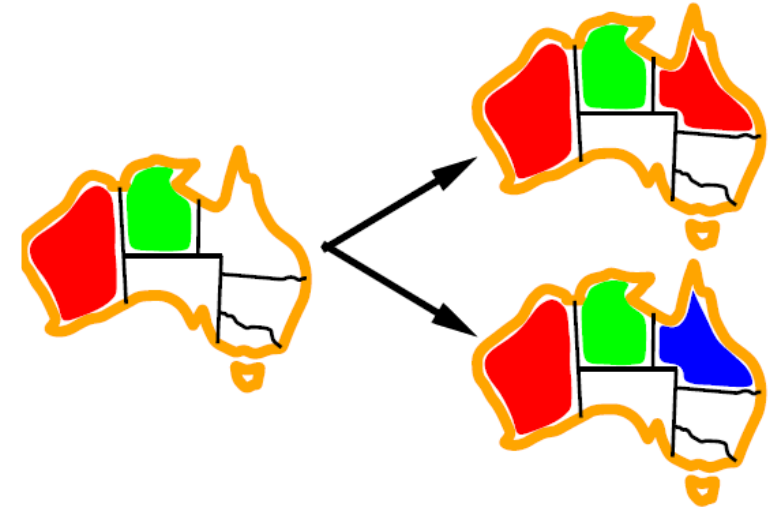
Ordering: Minimum Remaining Values

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



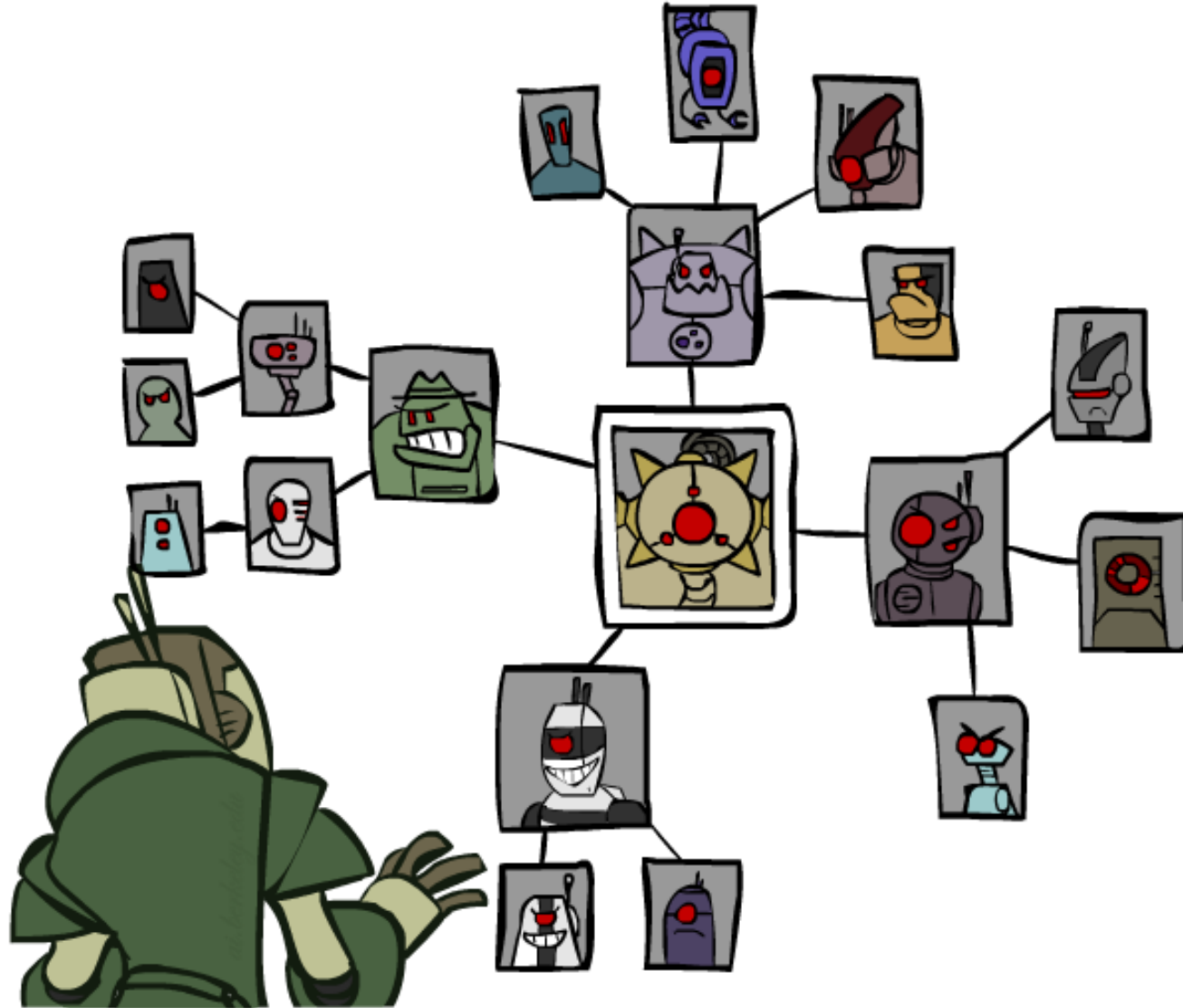
Ordering: Least Constraining Value

- 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)
- Combining these ordering ideas makes 1000 queens feasible



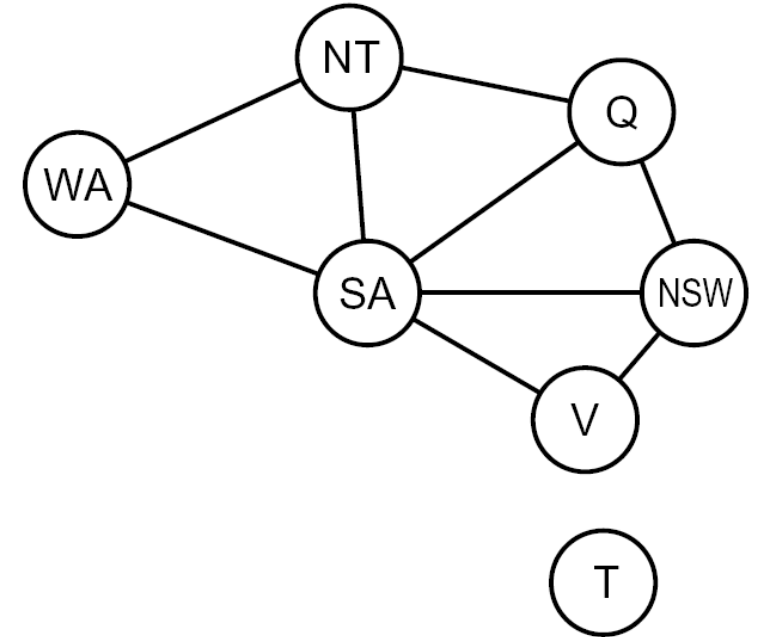
Demo -- Backtracking + Forward Checking + Ordering

Structure

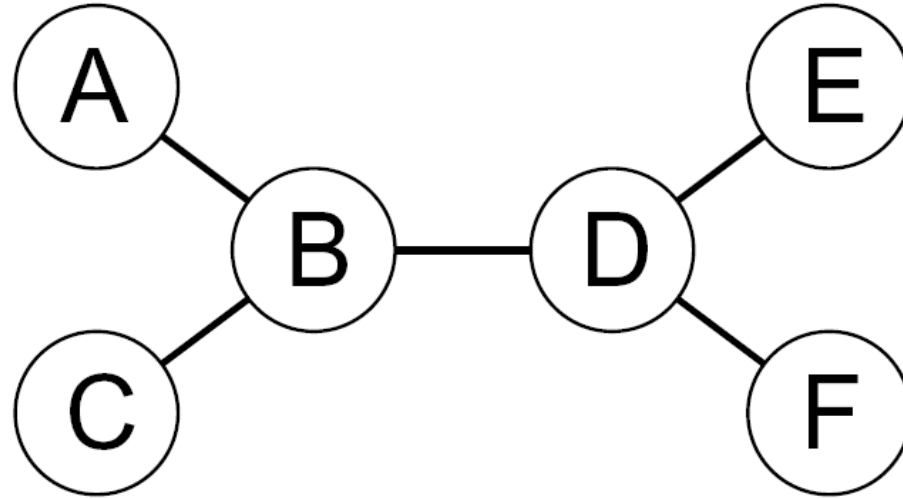


Problem Structure

- 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



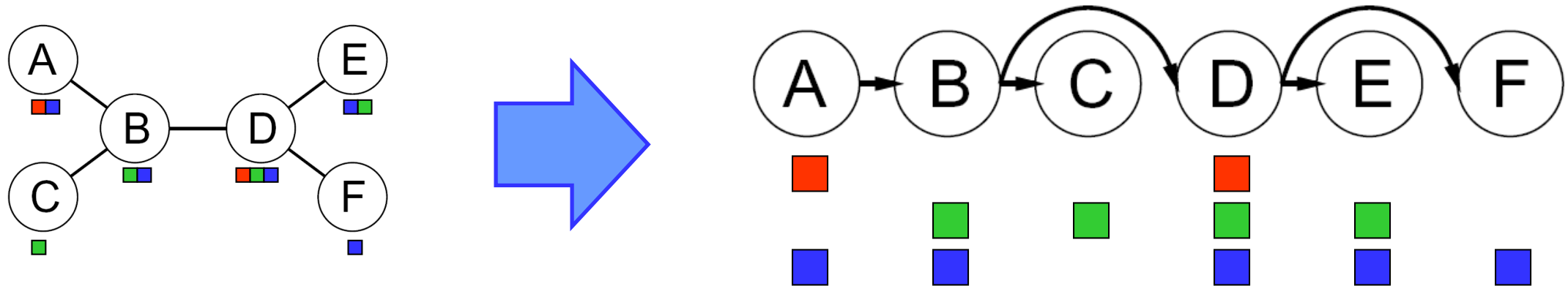
Tree-Structured CSPs



- 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

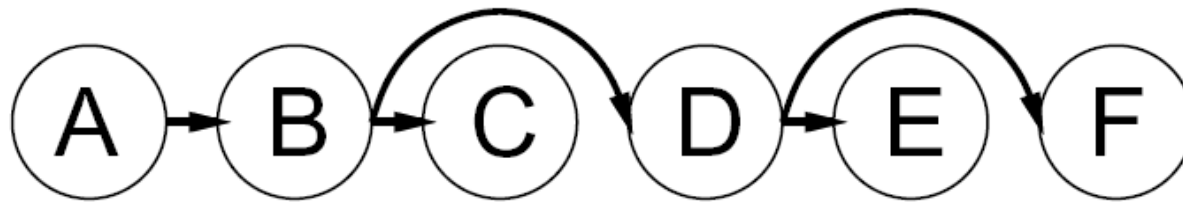
- Algorithm for tree-structured CSPs:
 - Order: Choose a root variable, order variables so that parents precede children



- 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: $O(n d^2)$

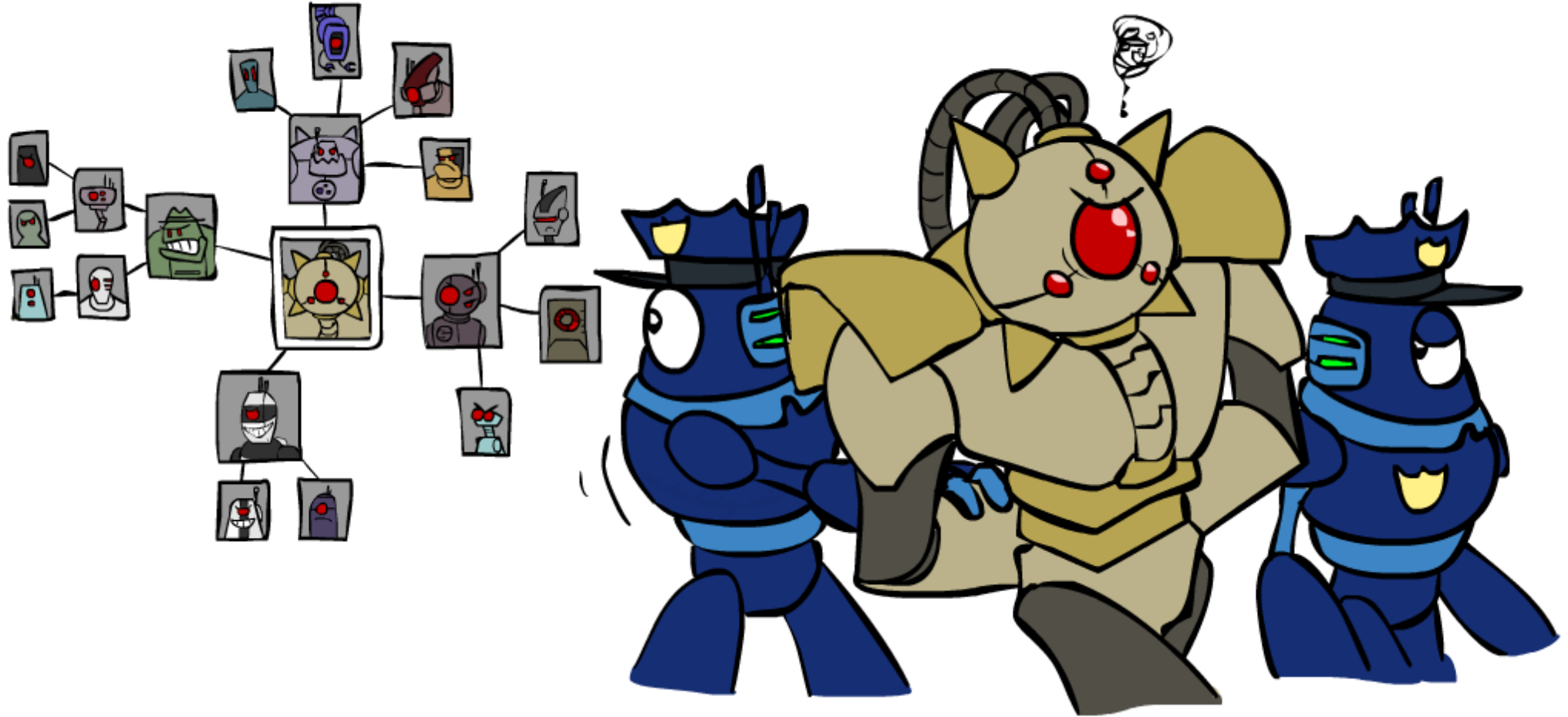
Tree-Structured CSPs

- 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)

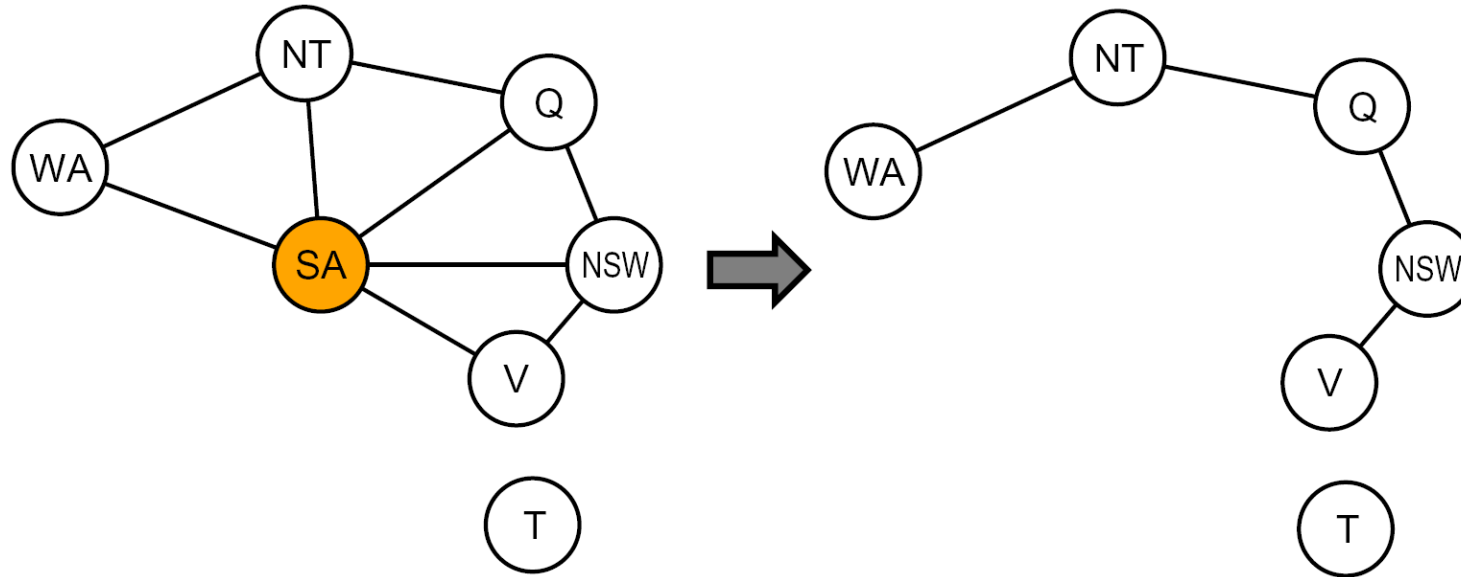


- Claim 2: If root-to-leaf arcs are consistent, forward assignment will not backtrack
- Easy to prove
- Why doesn't this algorithm work with cycles in the constraint graph?
- Note: we'll see this basic idea again with Bayes' nets

Cutset Conditioning



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

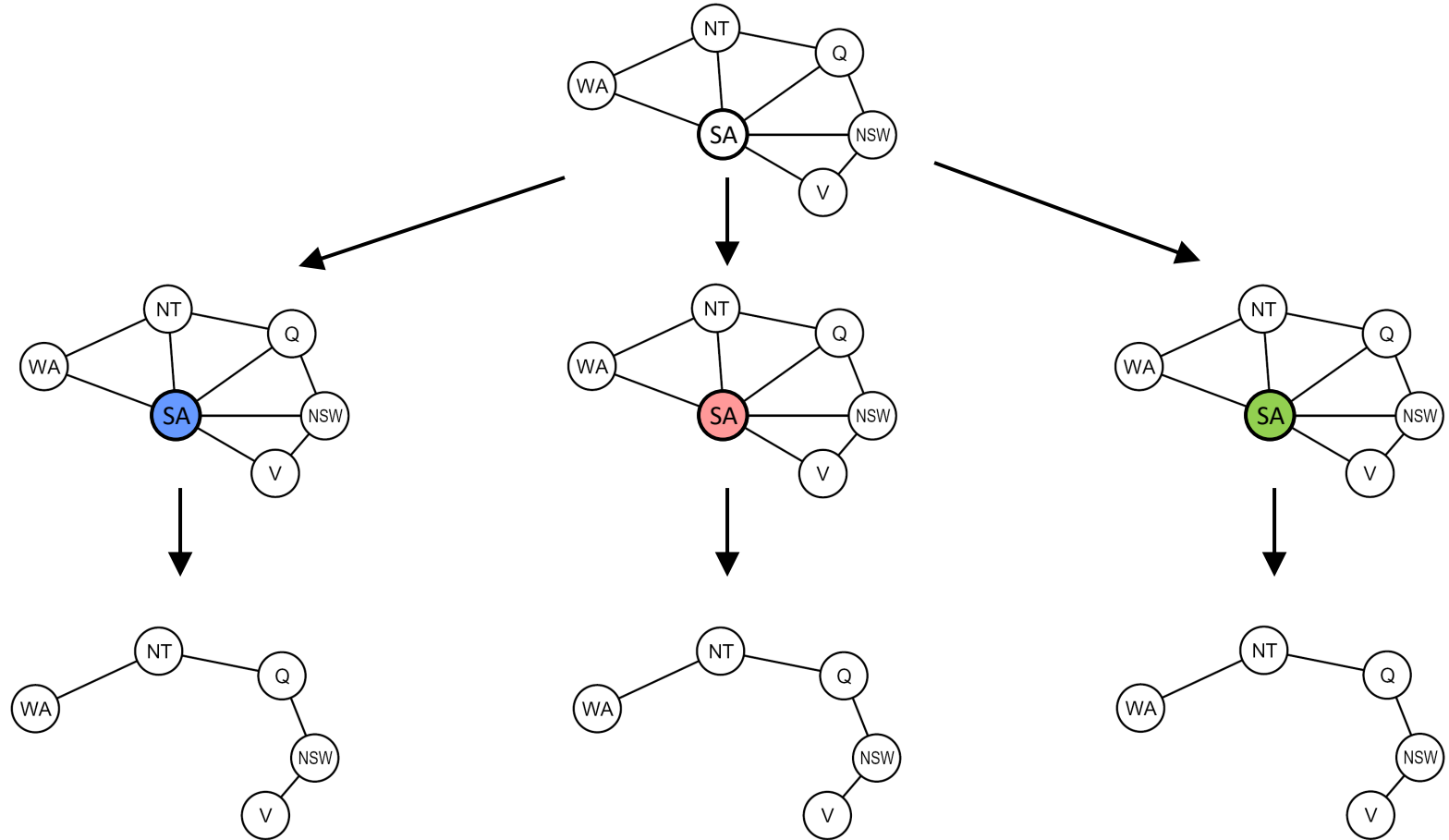
Cutset Conditioning

Choose a cutset

Instantiate the cutset
(all possible ways)

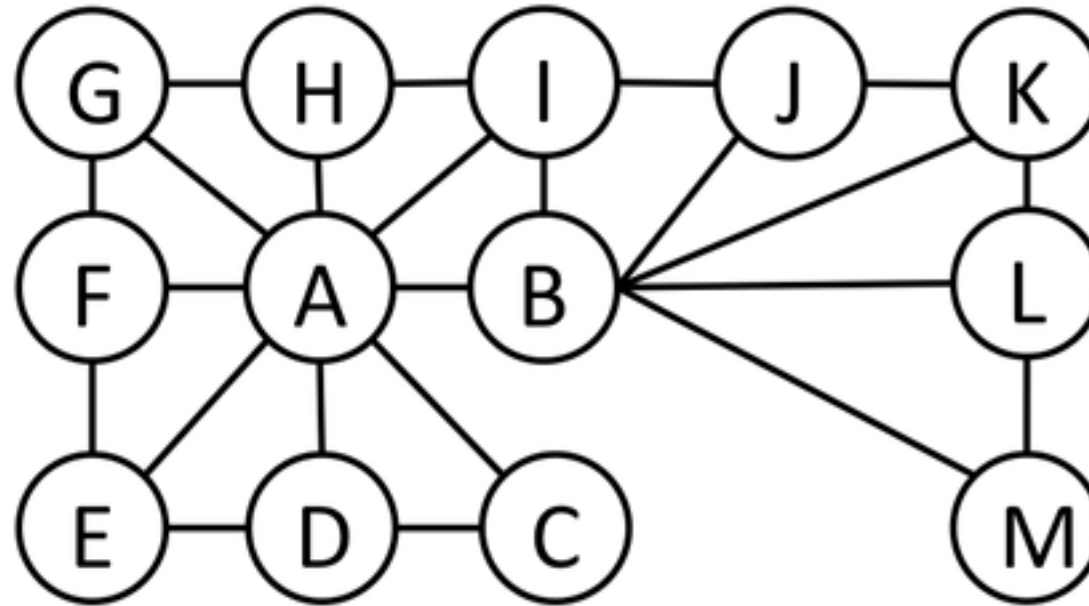
Compute residual CSP
for each assignment

Solve the residual CSPs
(tree structured)



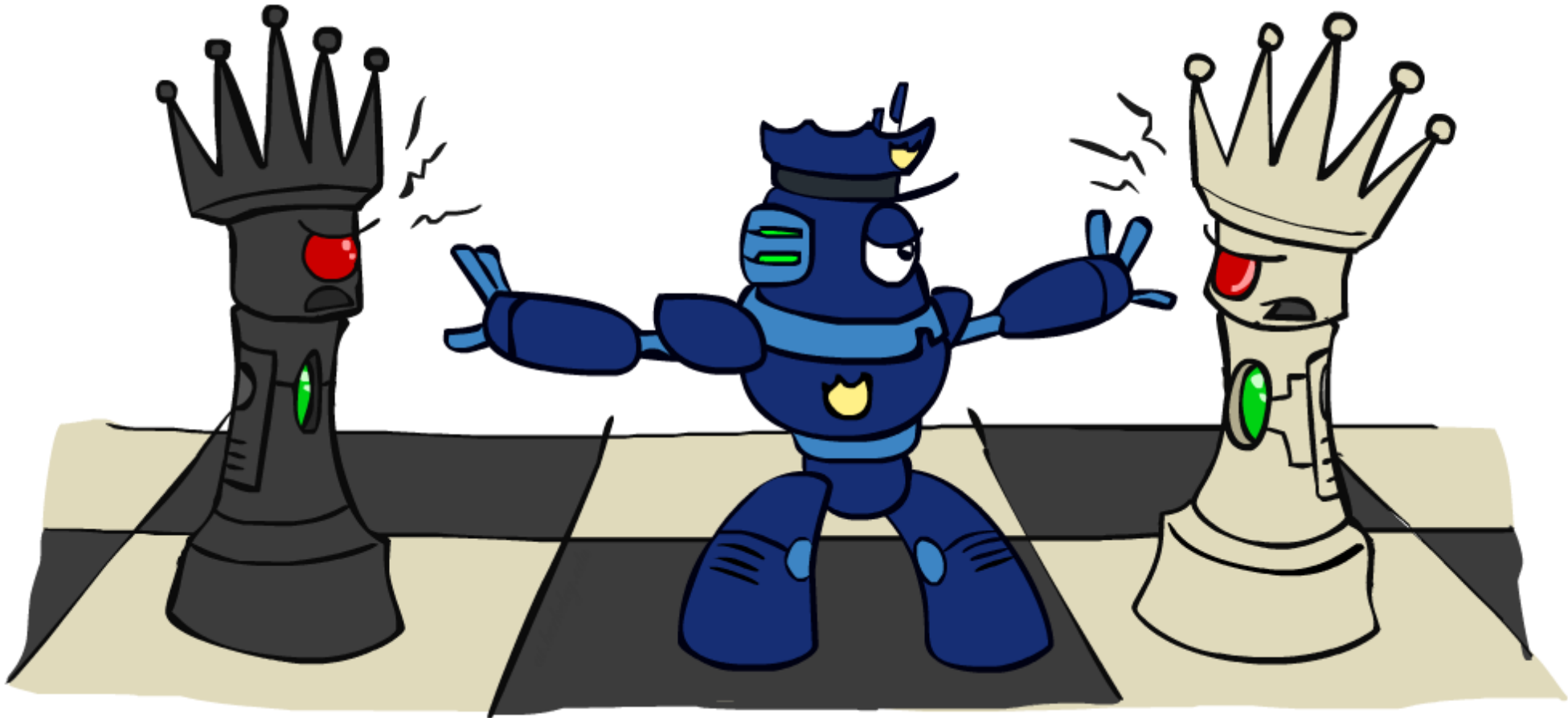
Finding Cutset

- Find the smallest cutset for the graph below.



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

Iterative Improvement



Iterative Algorithms for CSPs

- Idea:

- Take a complete assignment with unsatisfied constraints
- *Reassign* variable values to minimize 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

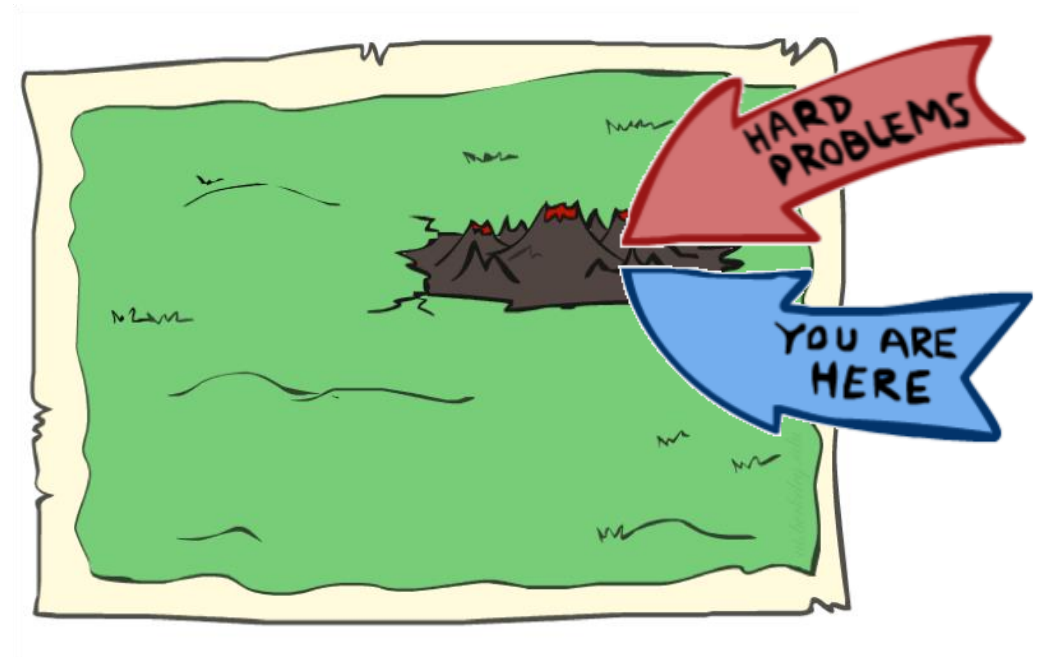
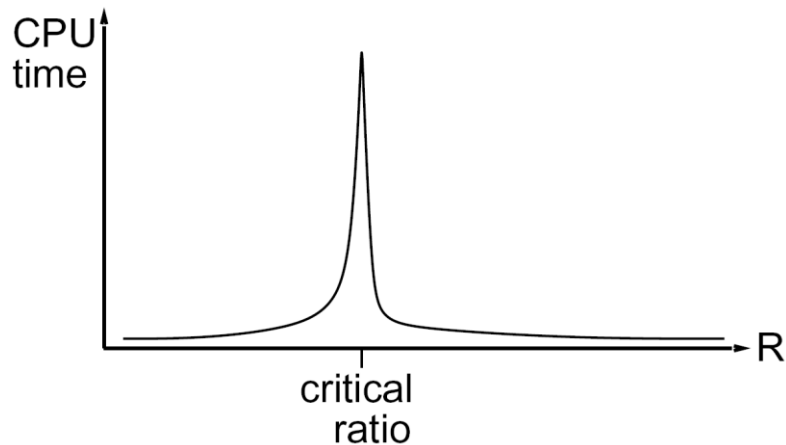


Demo – Iterative Improvement – Coloring

Performance

- 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

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

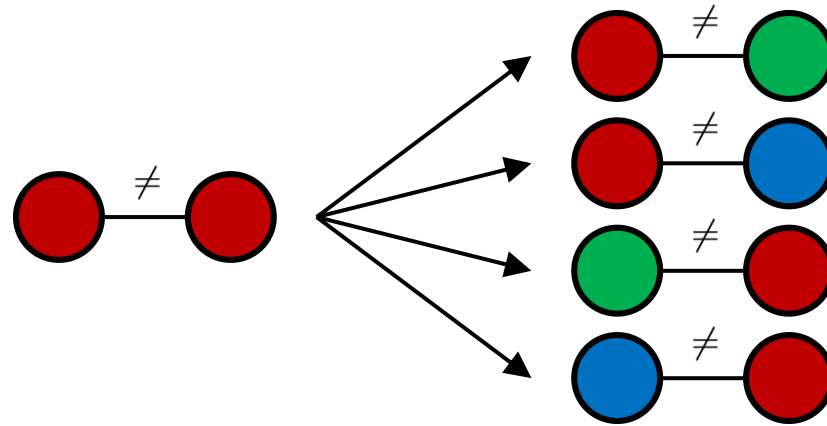


Local Search



Local Search

- Goal: identification, optimization
- Local search: improve a single option until you can't make it better
- State: a complete assignment
- Successor function: local changes



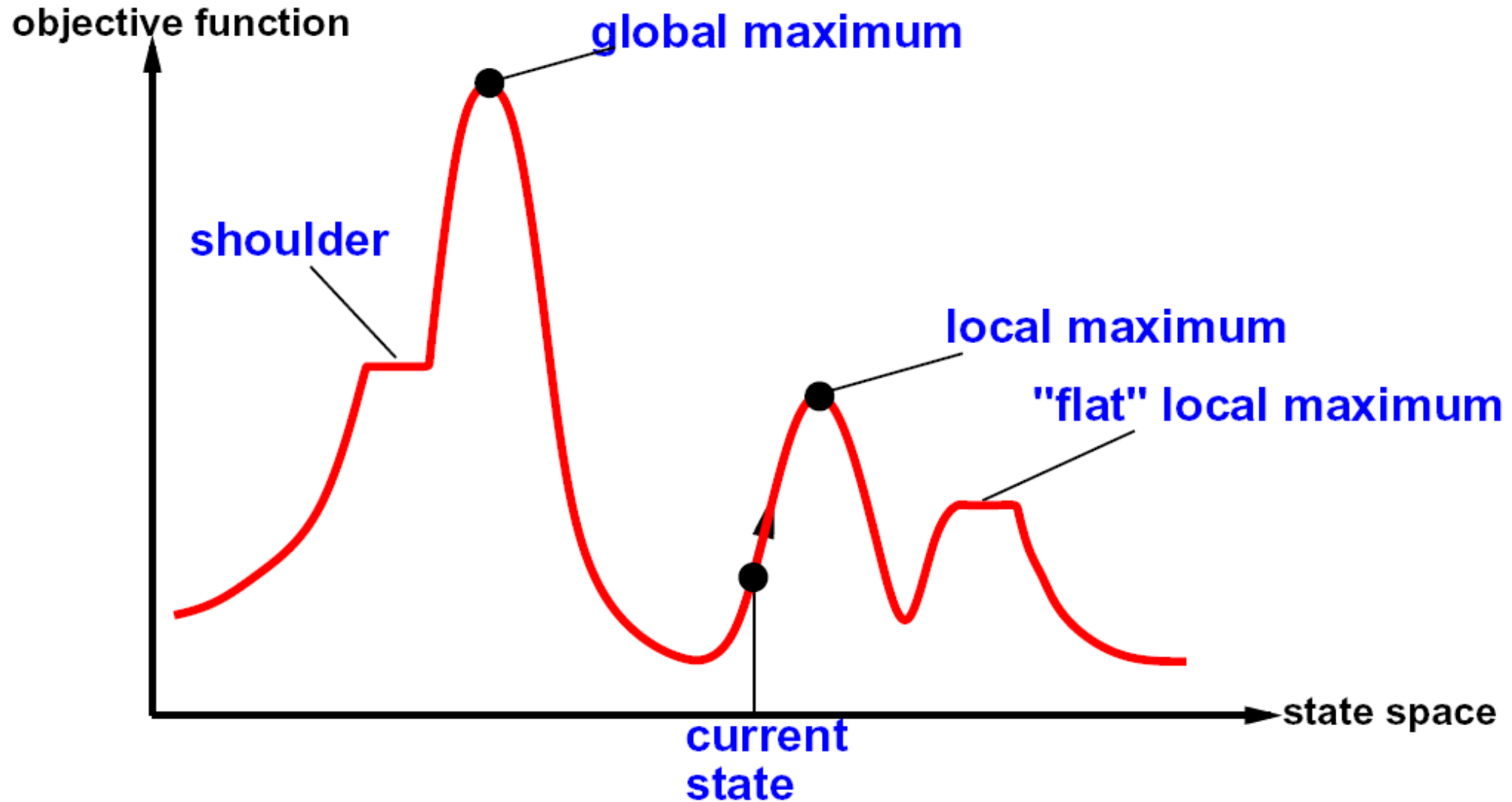
- Generally much faster and more memory efficient (but incomplete and suboptimal)

Hill Climbing

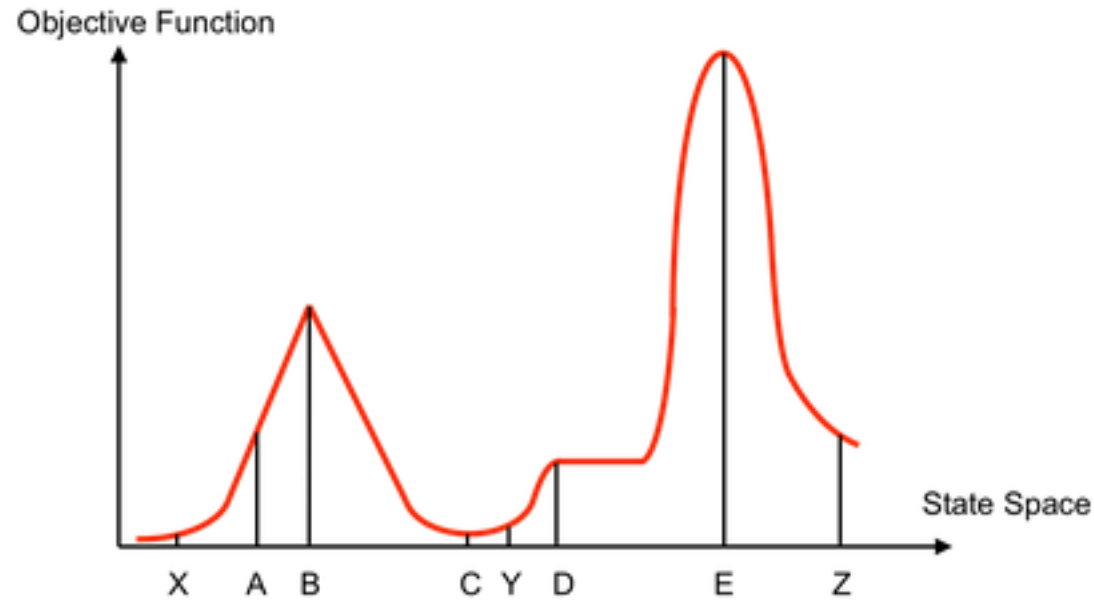
- Simple, general idea:
 - Start wherever
 - Repeat: move to the best neighboring state
 - If no neighbors better than current, quit
- What's good about this approach?
 - Simple, fast
- What's bad about it?



Hill Climbing Diagram



Hill Climbing Quiz



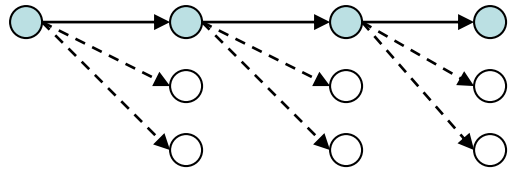
Starting from X, where do you end up ?

Starting from Y, where do you end up ?

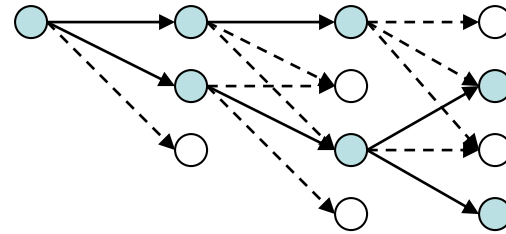
Starting from Z, where do you end up ?

Beam Search

- Like greedy hill climbing search, but keep K states at all times:



Greedy Search

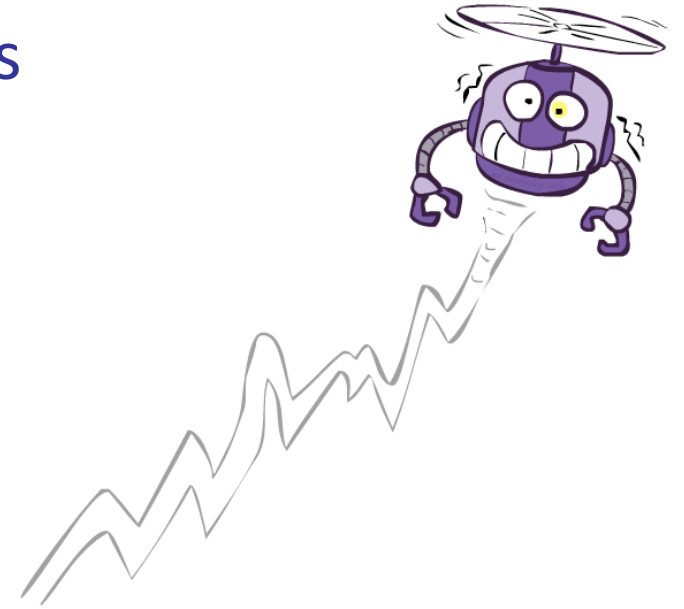


Beam Search

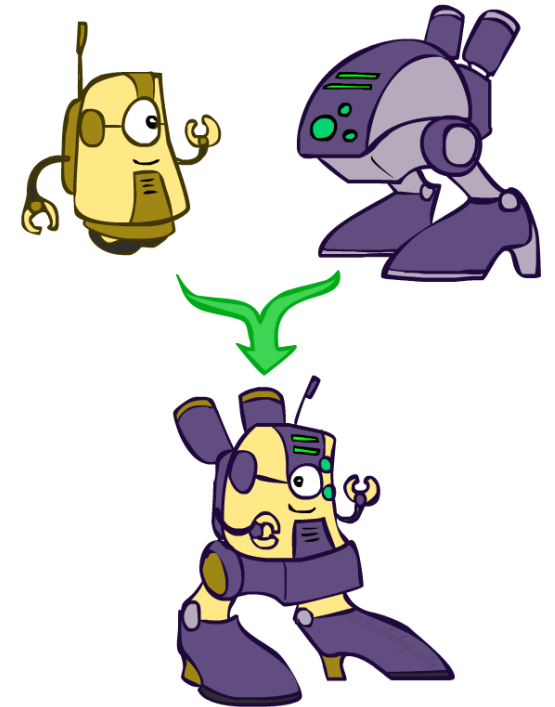
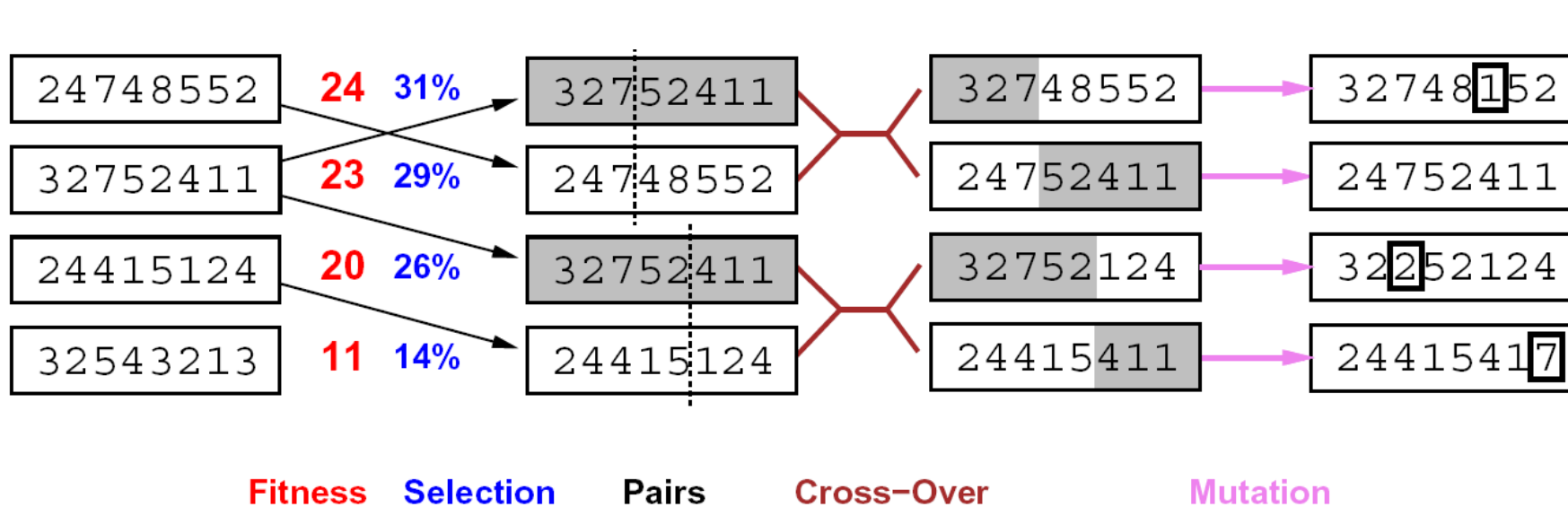
- The best choice in MANY practical settings
- Optimal?

Simulated Annealing

- Idea: Escape local maxima by allowing downhill moves
 - Pick a random move
 - Always accept an uphill move
 - Accept a downhill move with probability $e^{-\Delta E / T}$
 - But make the probability smaller (by decreasing T) as time goes on
- Theoretical guarantee
 - If T decreased slowly enough, will converge to optimal state!
- 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



Genetic Algorithms



- Genetic algorithms use a natural selection metaphor
 - Keep the best (or sample) N states at each step based on a fitness function
 - Pairwise crossover operators, with optional mutation to give variety

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:
 - Filtering
 - Forward Checking, Arc Consistency
 - Ordering
 - MRV, LCV
 - Structure
 - Tree structured, Cutset conditioning
- Iterative min-conflicts (local search) is often effective in practice

