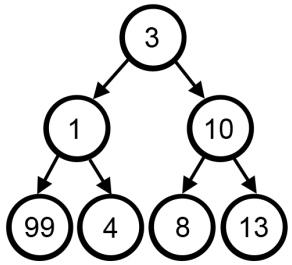
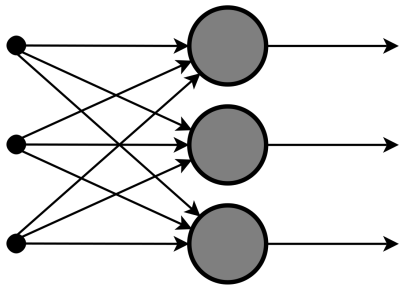
CSE 40171:
Artificial Intelligence



Constraint Satisfaction Problems: Local Search and Problem Structure

Homework #3 has been released
It is due at 11:59PM on 10/9

Local vs. Global Search: What are the advantages and disadvantages?

Local Search

Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Local Search

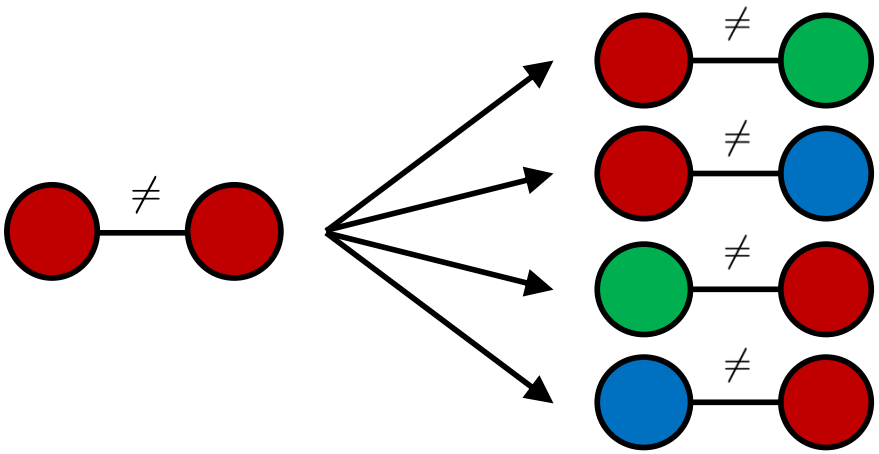
What if the path to the goal does matter?

Consider a class of algorithms that do not worry about paths at all

- Local search algorithms operate using a single current vertex
- Make moves only to neighbors of the current vertex

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



Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Two Key Advantages

1. They use very little memory — usually a constant amount

2. They can often find reasonable solutions in large or infinite (continuous) states spaces for which systematic algorithms are unsuitable

Two Possible Disadvantages

1. Incomplete
2. Suboptimal

Objective Functions

Local search algorithms are also useful for solving pure optimization problems

- Such problems aim to find the best state according to an objective function

Example:

$$\text{maximize or minimize } Z = \sum_{i=1}^{n} c_i X_i$$

c_i = the objective function coefficient corresponding to the i^{th} variable, and

X_i = the i^{th} decision variable.

State-Space Landscape

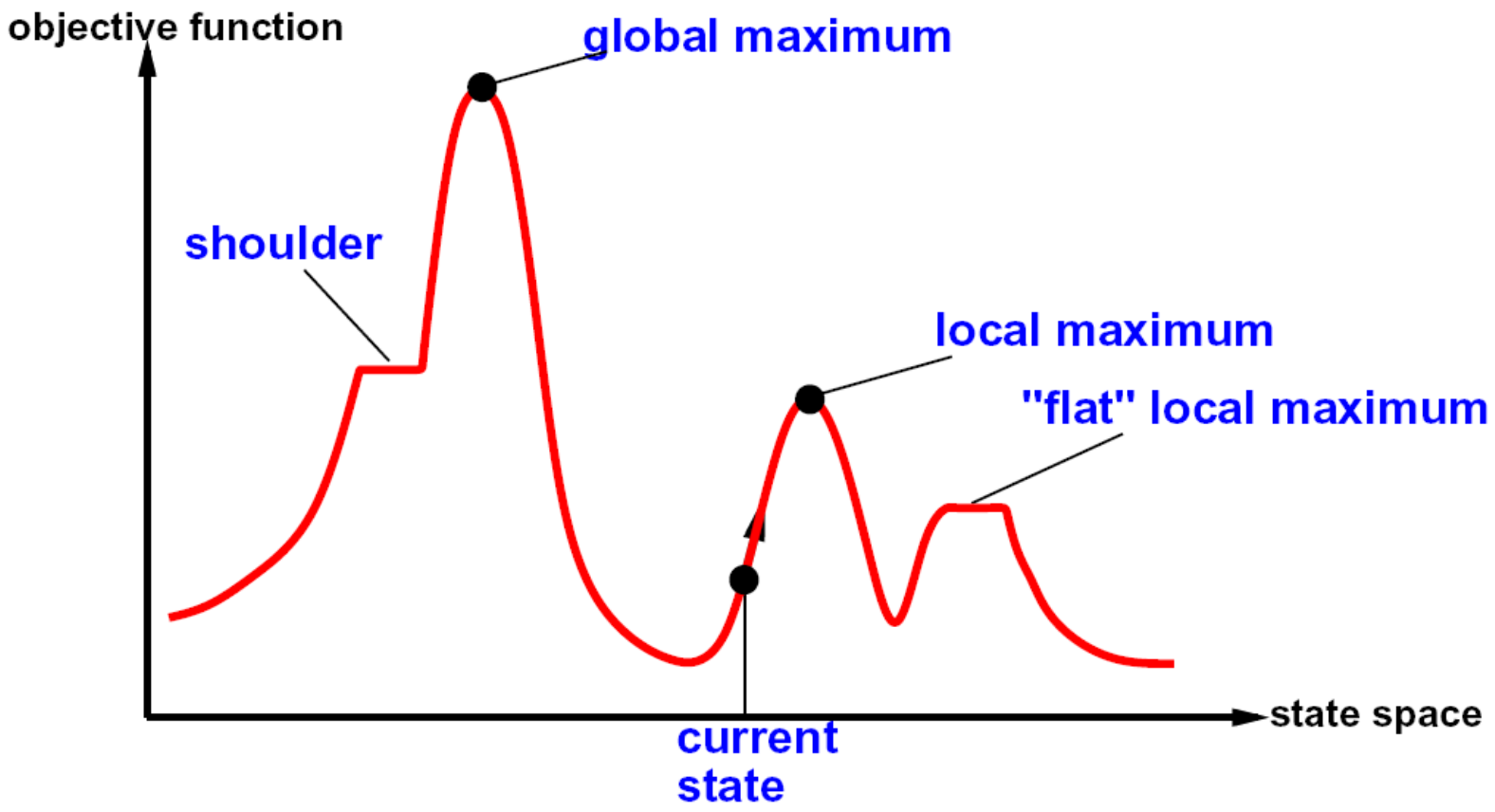


Image Credit: Russel and Norvig

Hill Climbing

Simple, general idea:

- Start wherever
- Repeat: move to the best neighboring state
- If no neighbors better than current, quit

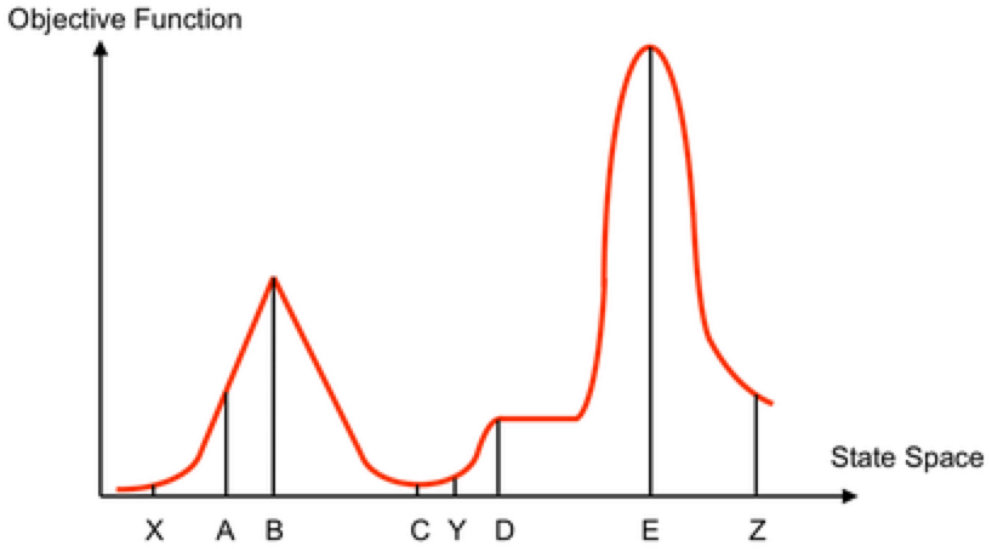
What's bad about this approach?

- Complete?
- Optimal?

What's good about it?

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Hill Climbing Quiz



1. Starting from X, where do you end up?

2. Starting from Y, where do you end up?

3. Starting from Z, where do you end up?

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

function MIN-CONFLICTS(*csp*, *max_steps*) **returns** a solution or failure

inputs: *csp*, a constraint satisfaction problem

max_steps, the number of steps allows before giving up

current ← an initial complete assignment for *csp*

for *i* = 1 to *max_steps* **do**

if *current* is a solution for *csp* **then return** *current*

var ← a randomly chosen conflicted variable from *csp*.VARIABLES

value ← the value *v* for *var* that minimizes CONFLICTS(*var*, *v*, *current*, *csp*)

set *var* = *value* in *current*

return *failure*

A two-step solution using min-conflicts

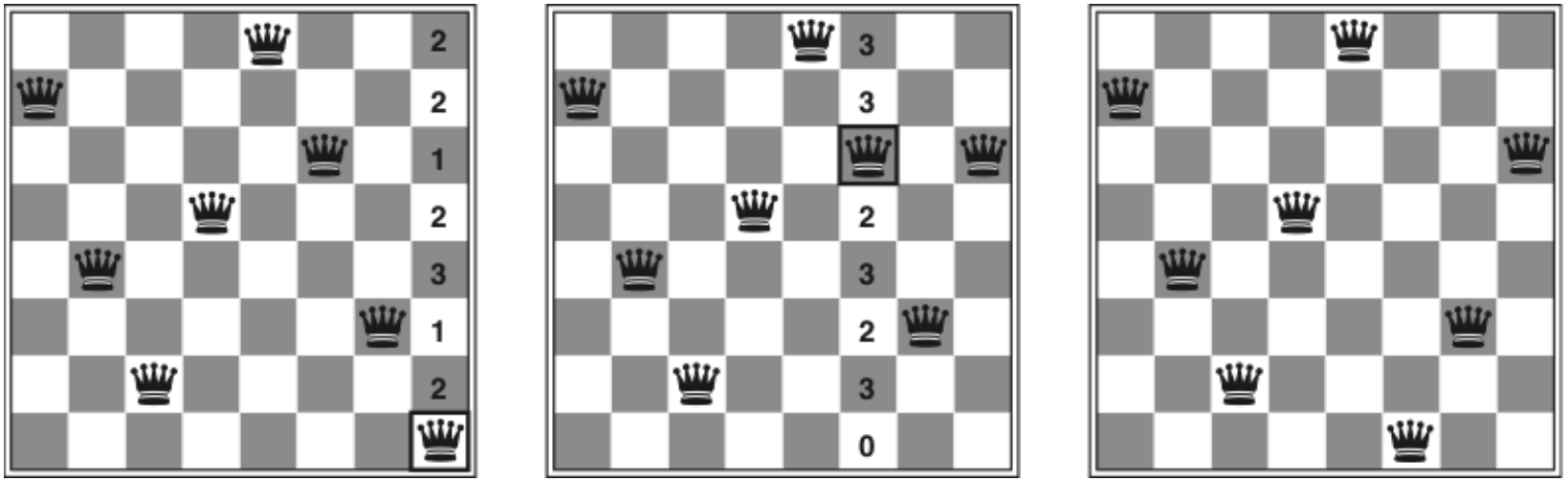
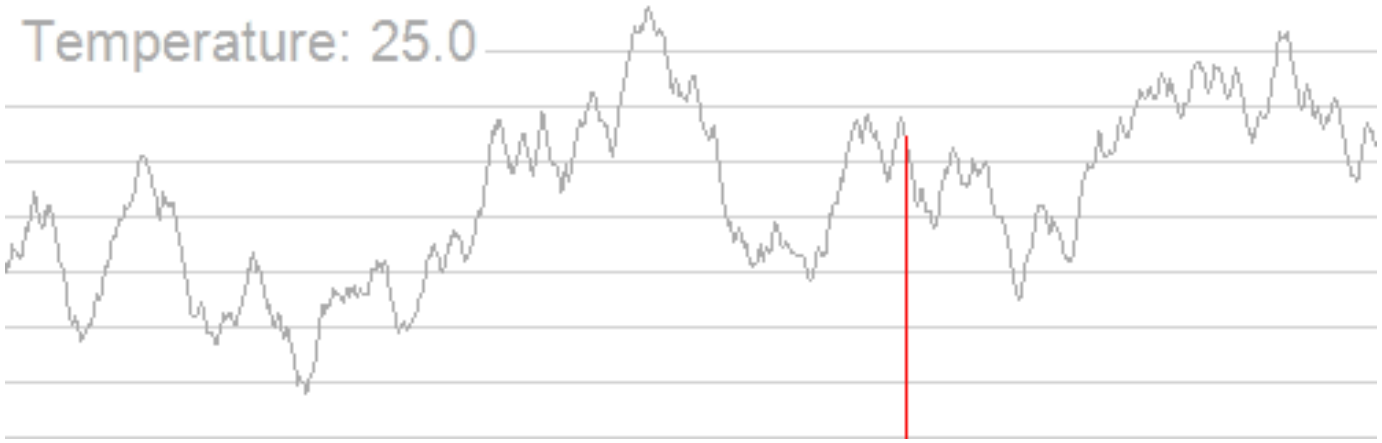


Image Credit: Russel and Norvig

Simulated Annealing



```
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 ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(Δ E/T)
```

Simulated Annealing

Theoretical guarantee: $p(x) \propto e^{\frac{E(x)}{kT}}$

- Stationary distribution:
- If T decreased slowly enough will converge to optimal state!

Is this an interesting guarantee?

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Simulated Annealing

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
- People think hard about ridge operators which let you jump around the space in better ways

Genetic Algorithms

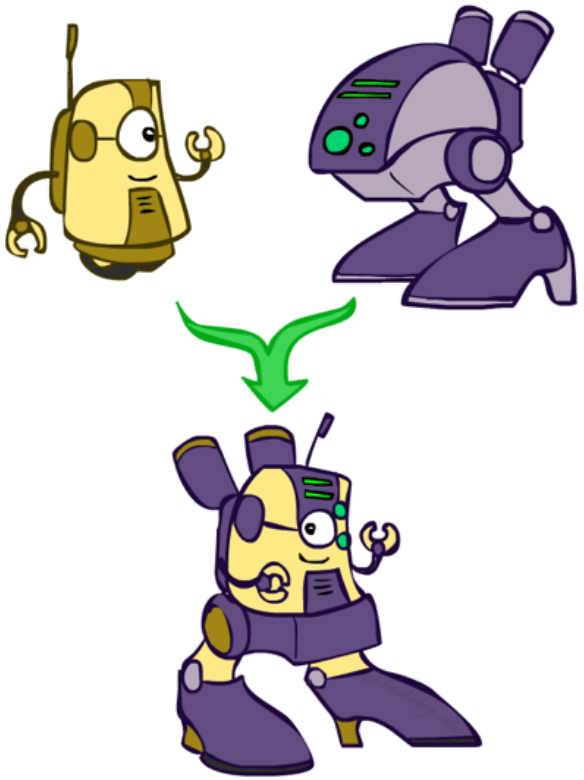


Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

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

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Genetic Algorithms

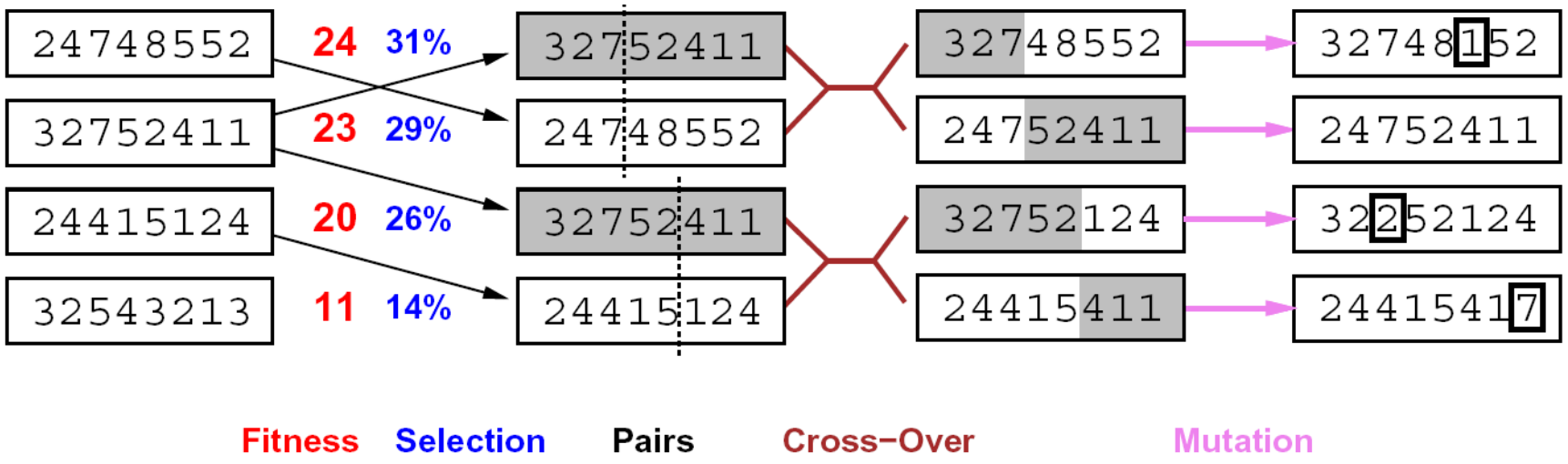
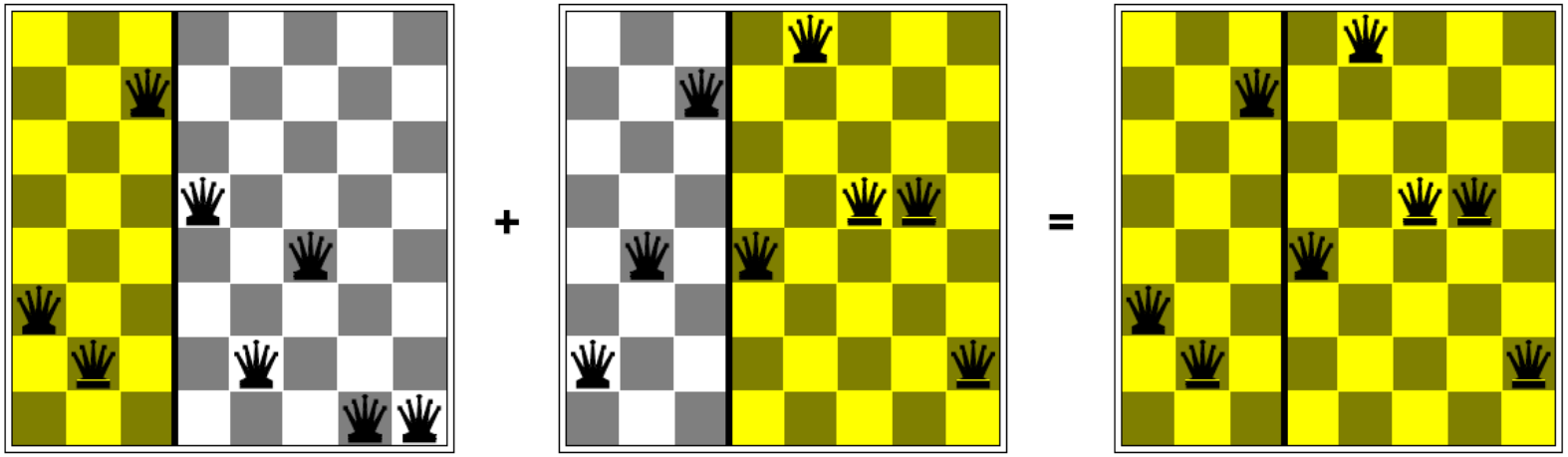


Image Credit: Russel and Norvig

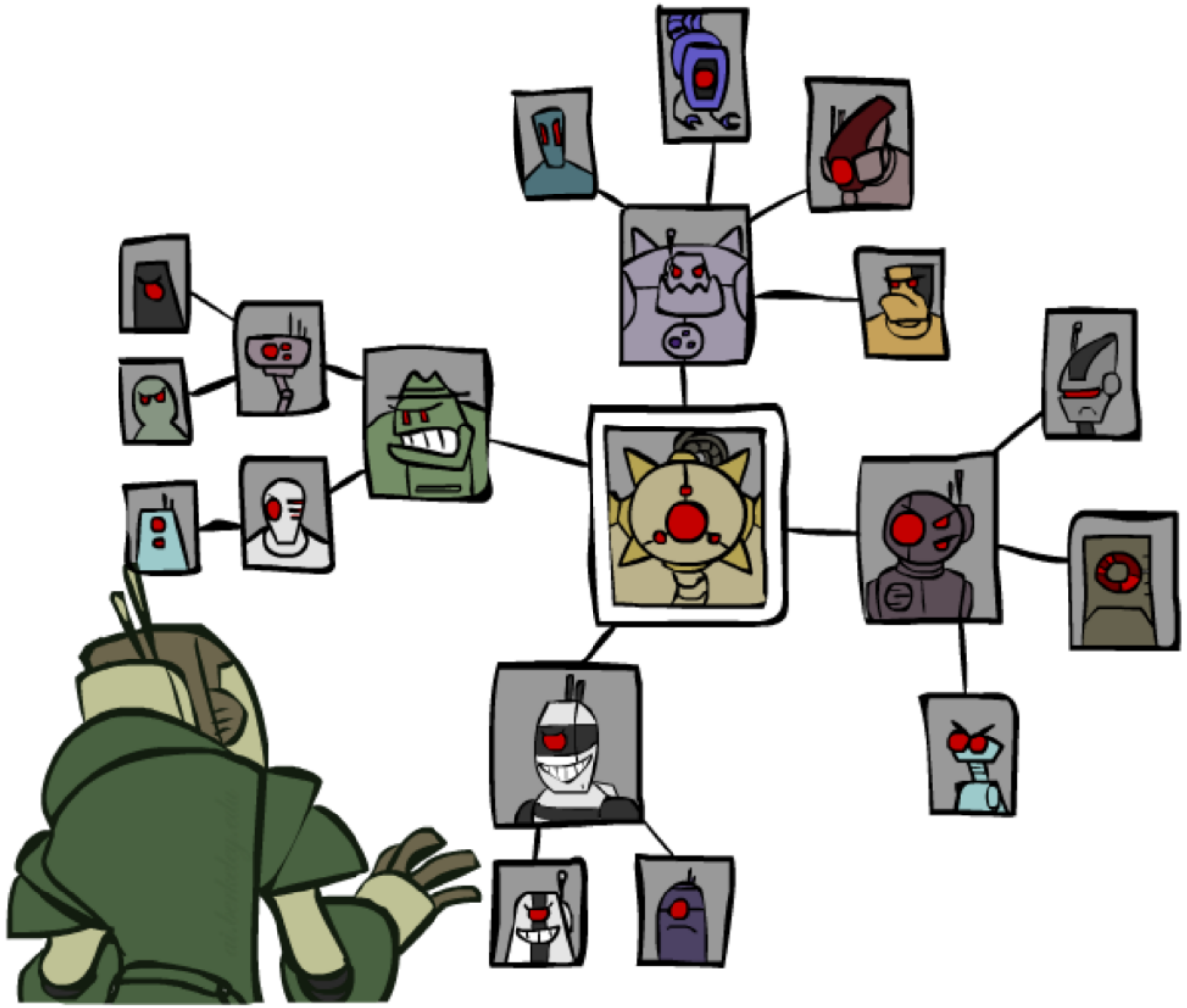
Example: N-Queens



1. Why does crossover make sense here?
2. When wouldn't it make sense?
3. What would mutation be?
4. What would a good fitness function be?

Image Credit: Russel and Norvig

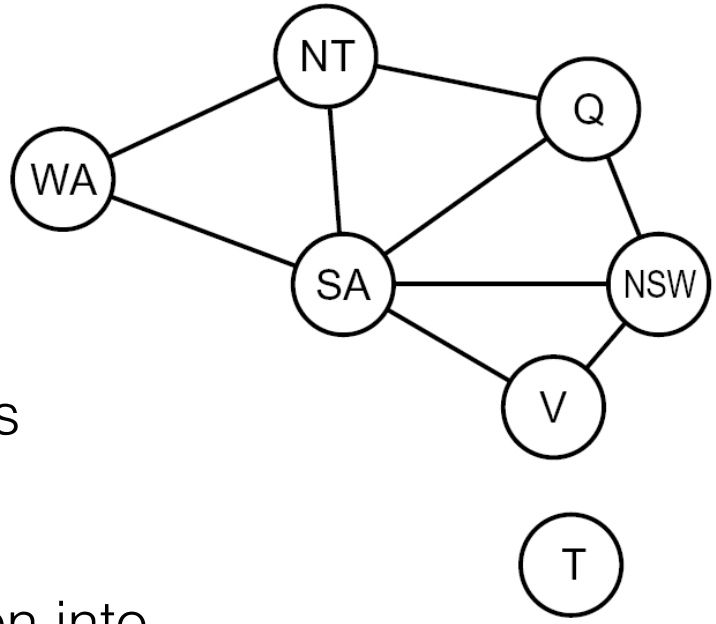
Structure



Problem Structure

Extreme case: independent subproblems

- Example: Tasmania and mainland do not interact



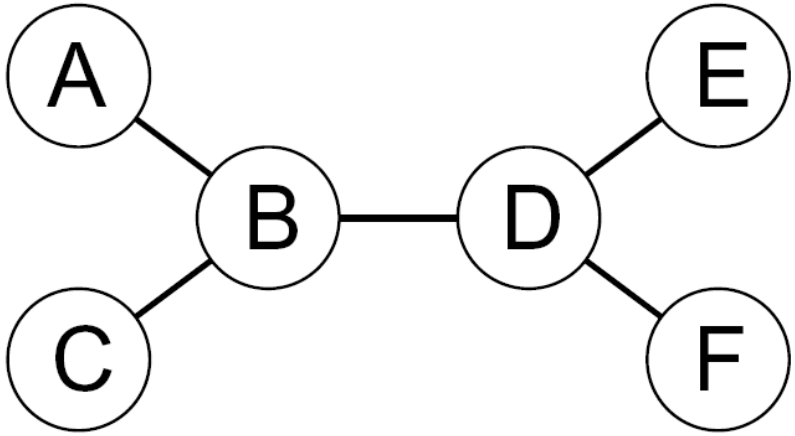
Independent subproblems are identifiable as connected components of constraint graph

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 vertices/sec
- $(4)(2^{20})$ = 0.4 seconds at 10 million vertices/sec

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Tree Structured CSPs



Theorem: if the constraint graph has no loops, the CSP can be solved in $O(nd^2)$ time

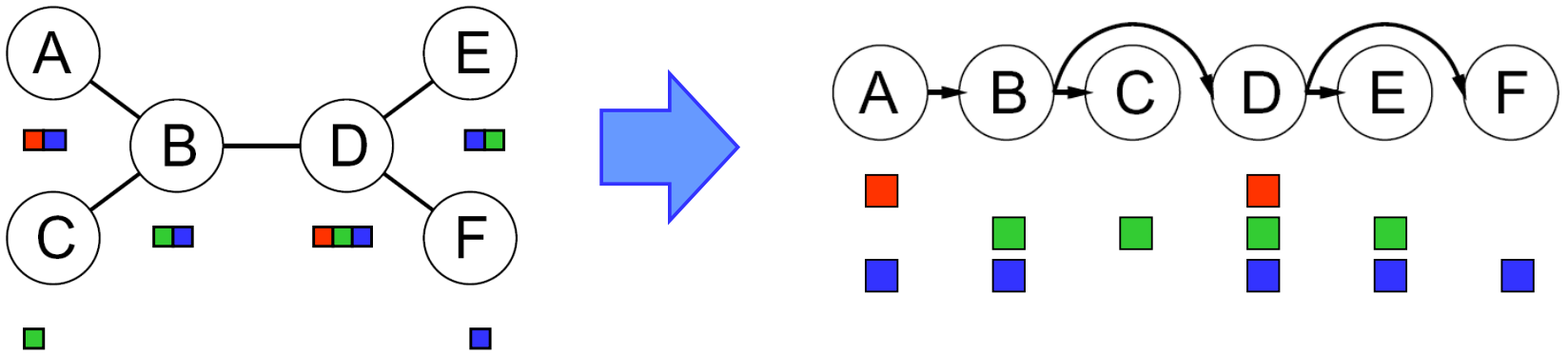
- Compare to general CSPs, where worst-case time is $O(d^n)$

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Tree Structured CSPs

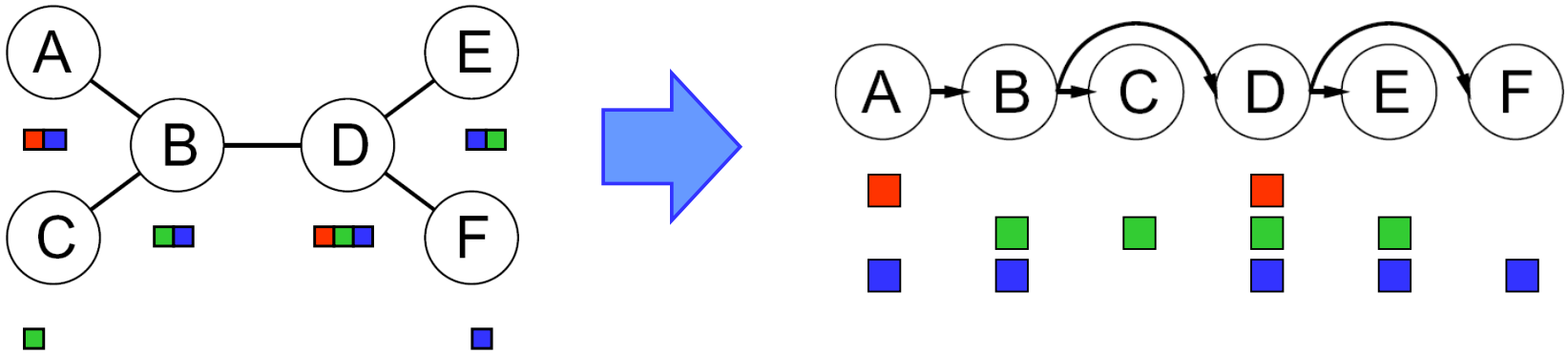
Algorithm for tree-structured CSPs:

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



Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Tree Structured CSPs



Remove backward: for $i = n : 2$, apply RemoveInconsistent(Parent(X_i), X_i)

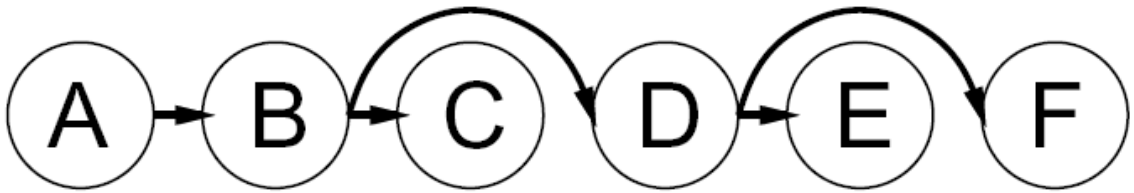
Assign forward: for $i = 1 : n$, assign X_i consistently with Parent(X_i)

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Tree Structured CSPs

Claim 1: After backward pass, all root-to-leaf edges 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)

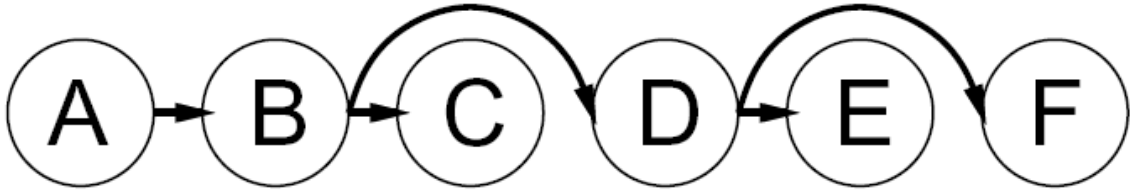


Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Tree Structured CSPs

Claim 2: If root-to-leaf edges are consistent, forward assignment will not backtrack

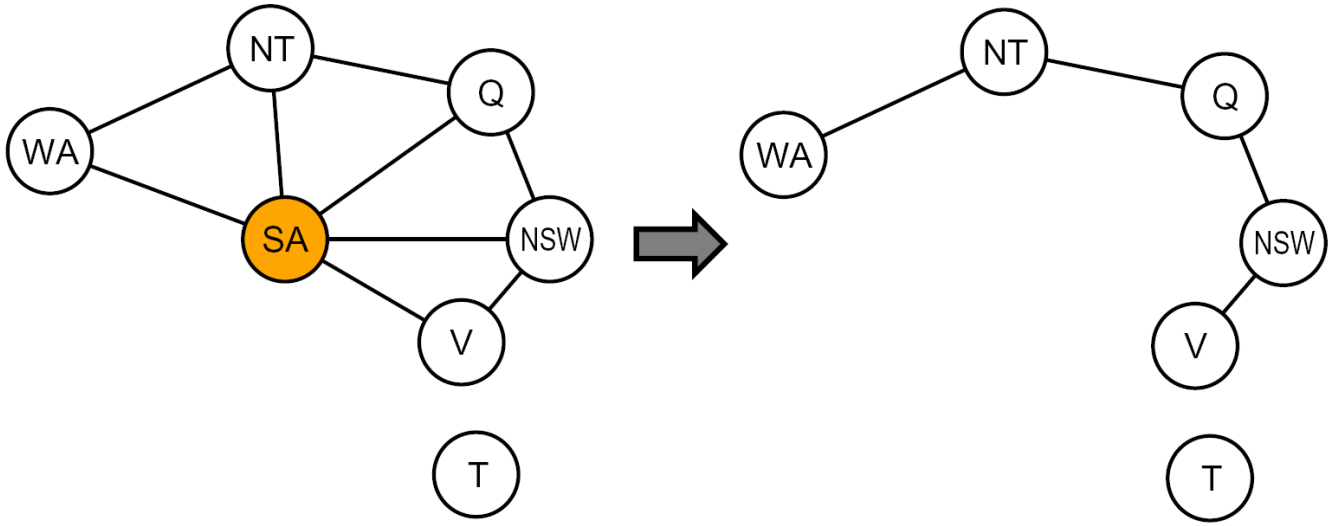
Proof: Induction on position



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

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

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

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

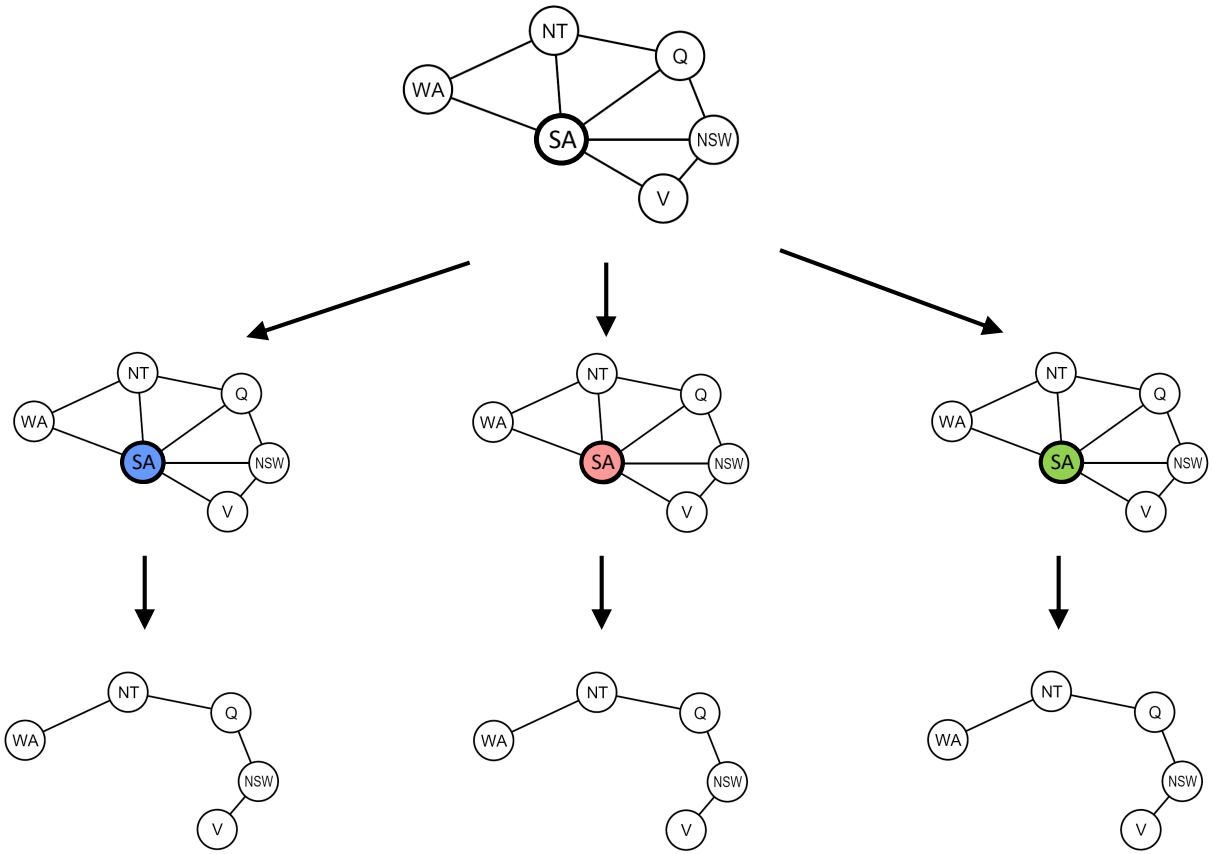
Cutset Conditioning

Choose a cutset

Instantiate the cutset (all possible ways)

Compute residual CSP for each assignment

Solve the residual CSPs (tree structured)

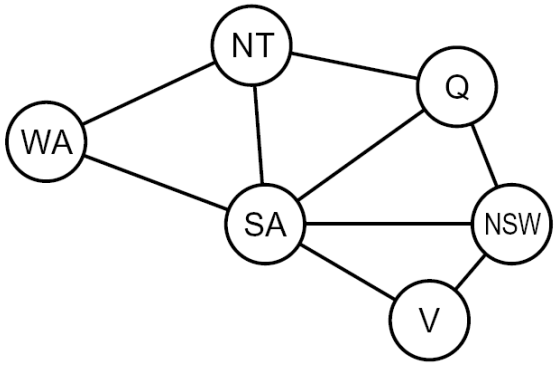


Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

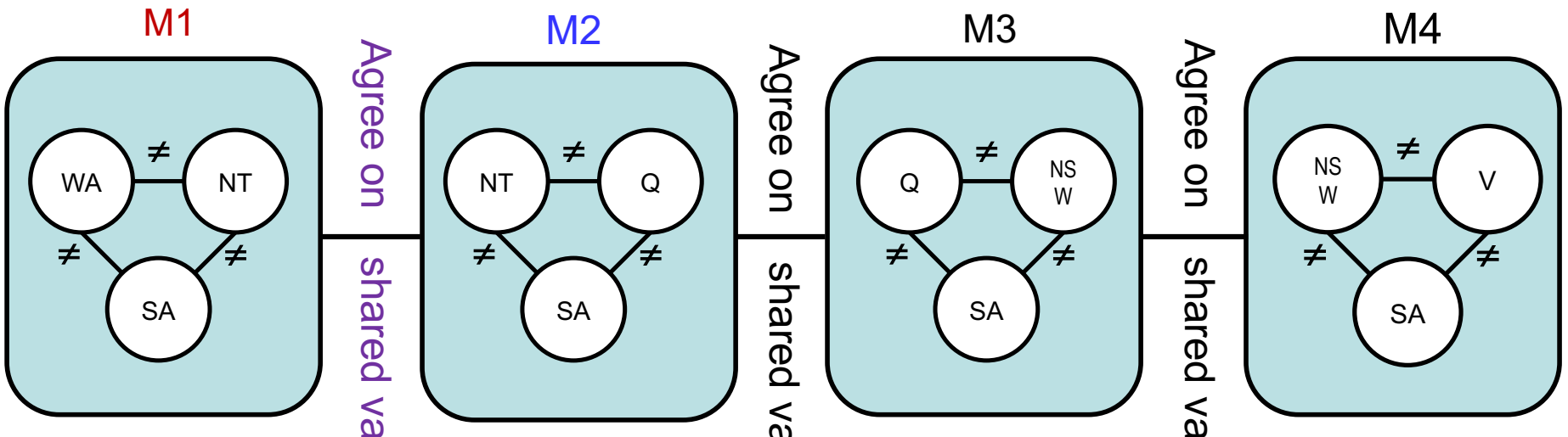
Tree Decomposition

Idea: create a tree-structured graph of mega-variables

Each mega-variable encodes part of the original CSP



Subproblems overlap to ensure consistent solutions



Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188