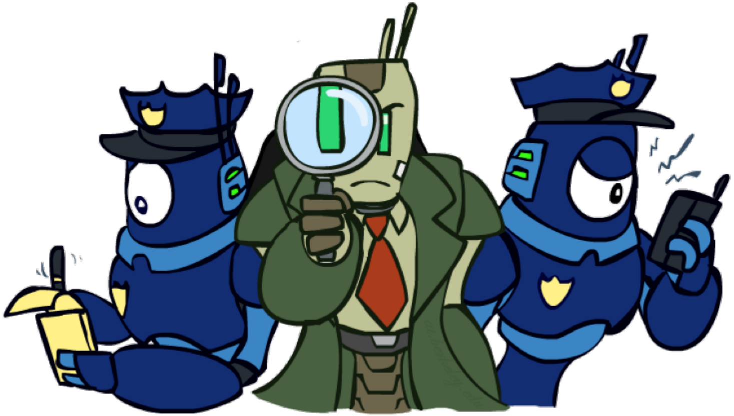
Announcement

- Homework 1: search, CSP, adversarial search
  - Available in Blackboard -> Homework
  - Due: March 8, 11:59pm

- Programming Assignment 1A
  - Submission at AutoLab
  - Due: Mar 5, 11:59pm

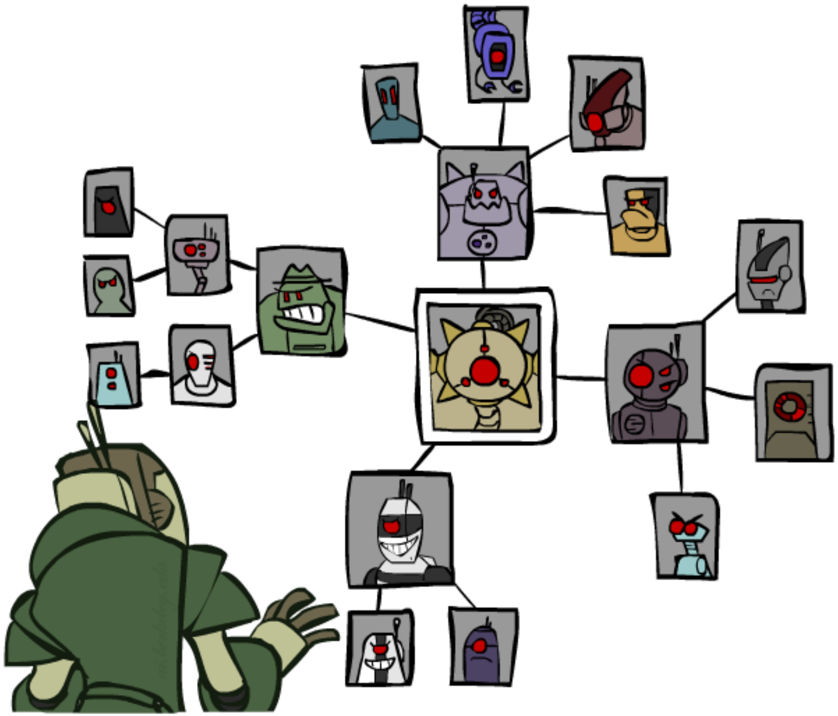
Constraint Satisfaction Problems



AIMA Chapter 6

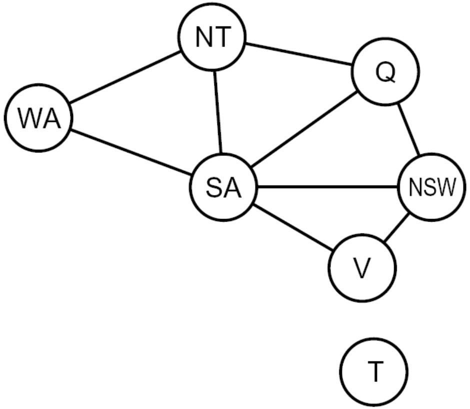
[Adapted from slides by Dan Klein and Pieter Abbeel at UC Berkeley (ai.berkeley.edu).]

Structure

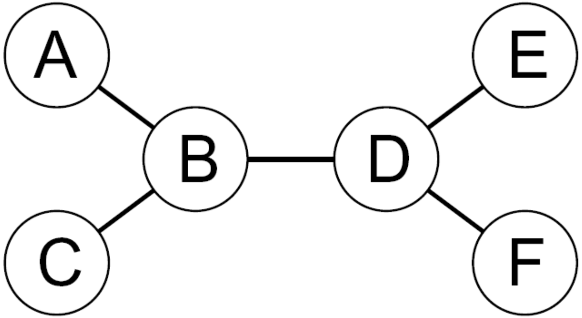


Problem Structure

- 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

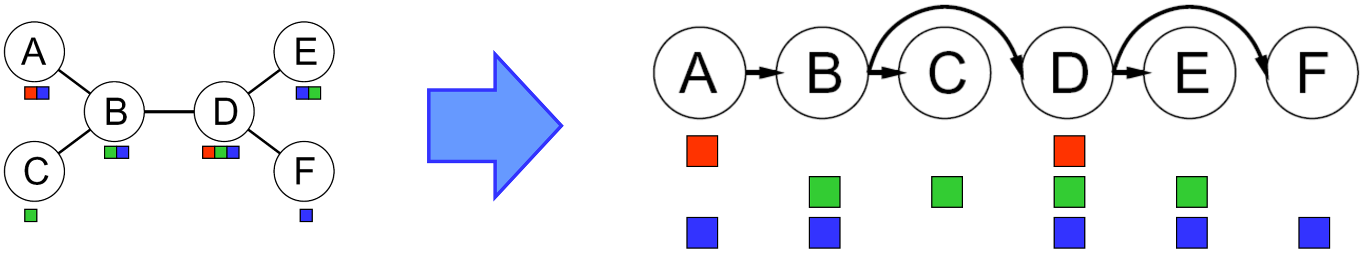


- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\ d^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$

- This property also applies to probabilistic reasoning (later)
- An example of the relation between syntactic restrictions and the complexity of reasoning

Tree-Structured CSPs

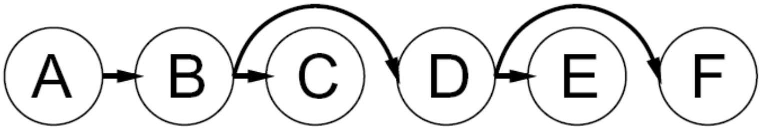
- Algorithm for tree-structured CSPs:
  - Order: Choose a root variable, order variables so that parents precede children



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

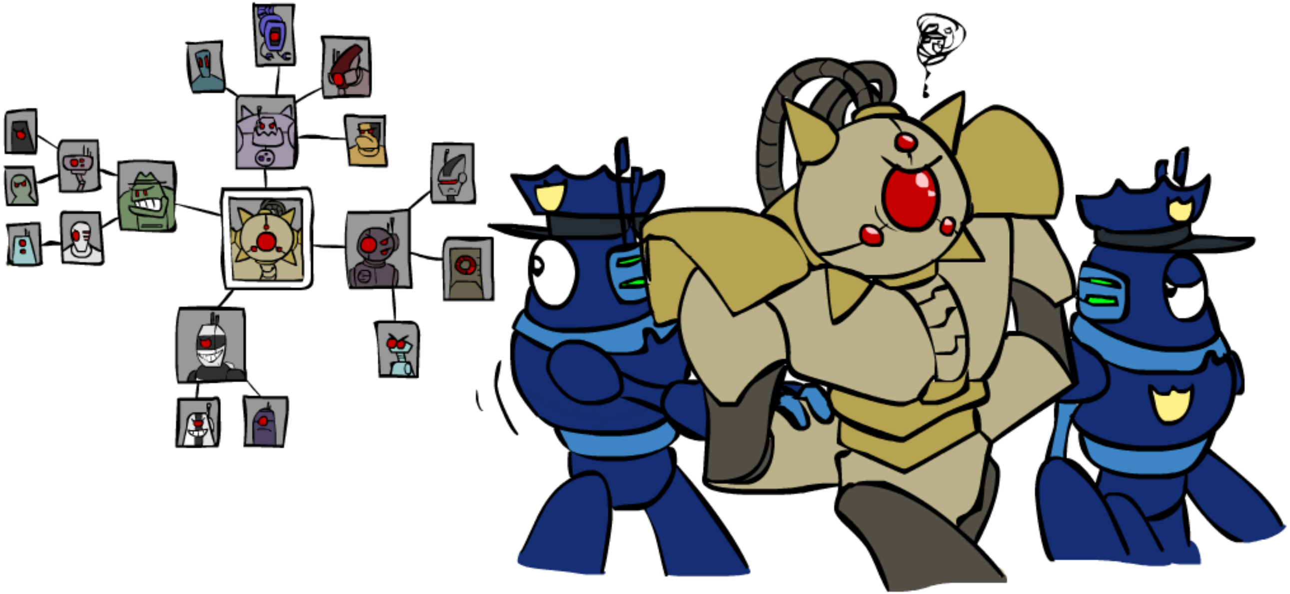


- Claim 2: If root-to-leaf arcs are consistent, forward assignment will not backtrack
- Easy to prove

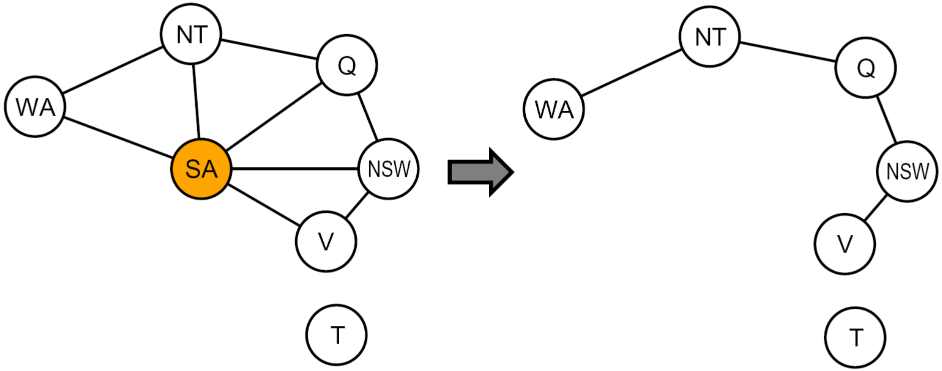
- Why doesn't this algorithm work with cycles in the constraint graph?

- 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

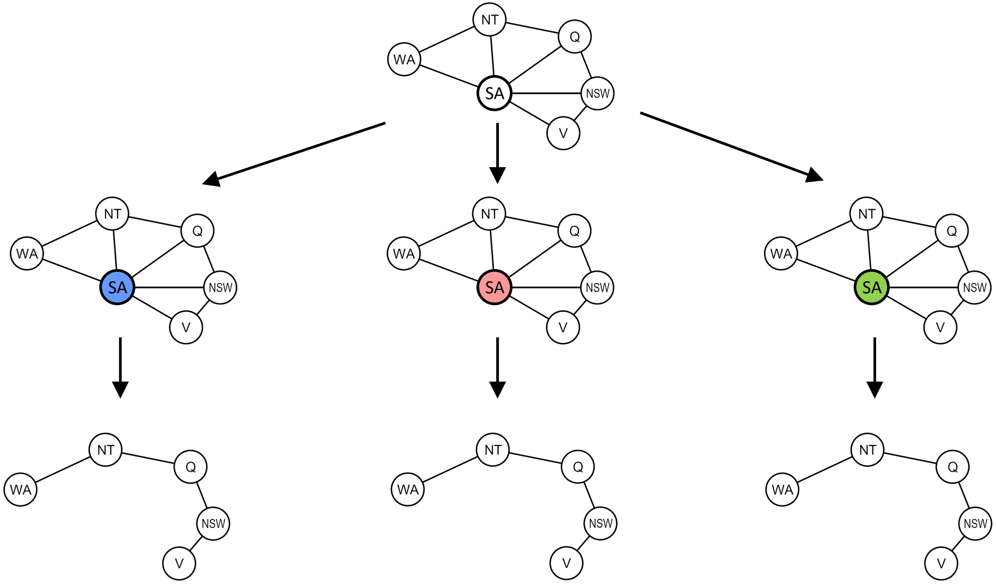
Cutset Conditioning

Choose a cutset

Instantiate the cutset (all possible ways)

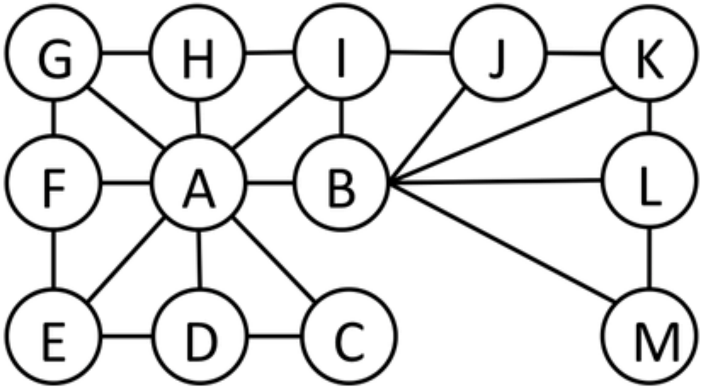
Compute residual CSP for each assignment

Solve the residual CSPs (tree structured)



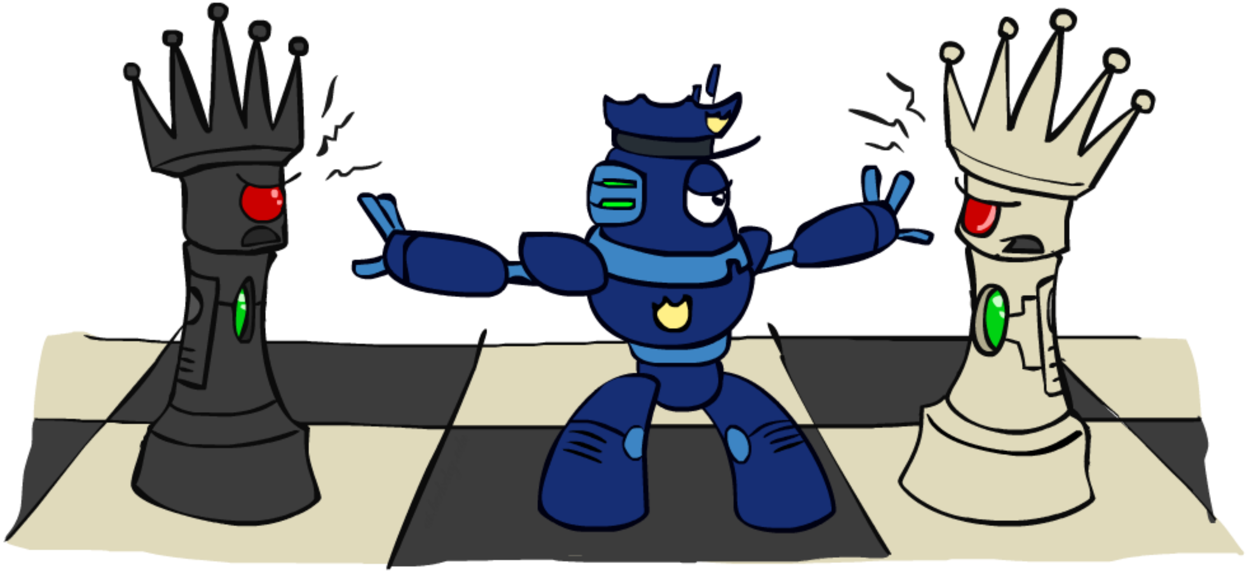
Finding Cutset

- Find the smallest cutset for the graph below.



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

Iterative Improvement

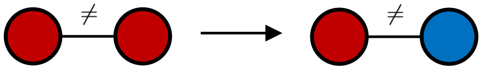


Iterative Algorithms for CSPs

CSP:

"Correct" (V.)

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

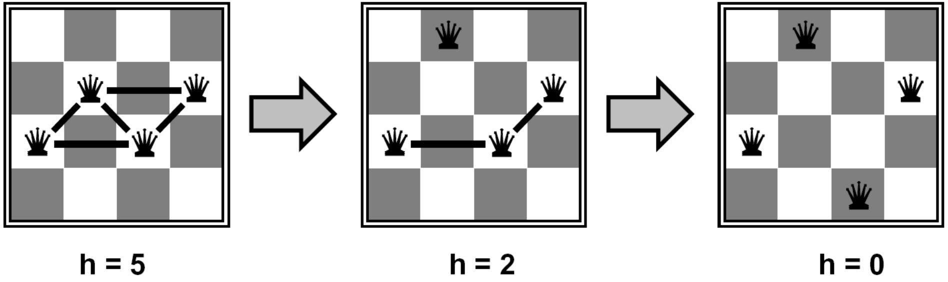


no fringe, no back-up plan.

- Algorithm: While not solved,
  - Variable selection: randomly select any conflicted variable
  - Value selection: min-conflicts heuristic:
    - Choose a value that violates the fewest constraints

re-assign to the fewest conflicts until finished.

Example: 4-Queens



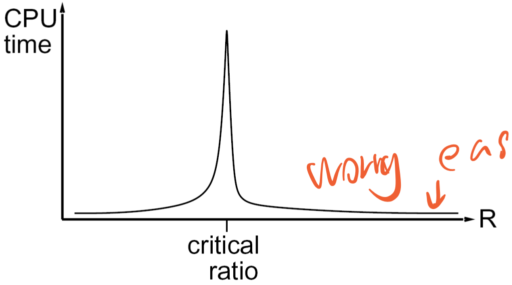
- States: 4 queens in 4 columns (4^4 = 256 states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: c(n) = number of attacks

Performance

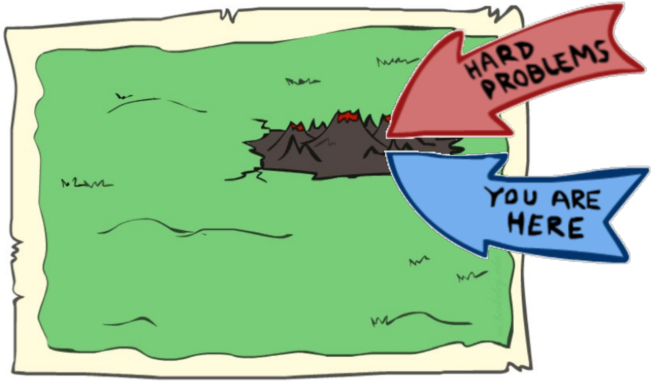
- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)!

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

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



wrong easy to detect



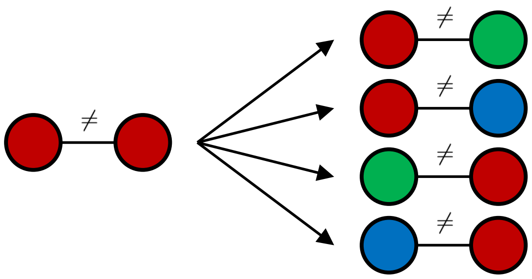
Local Search

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

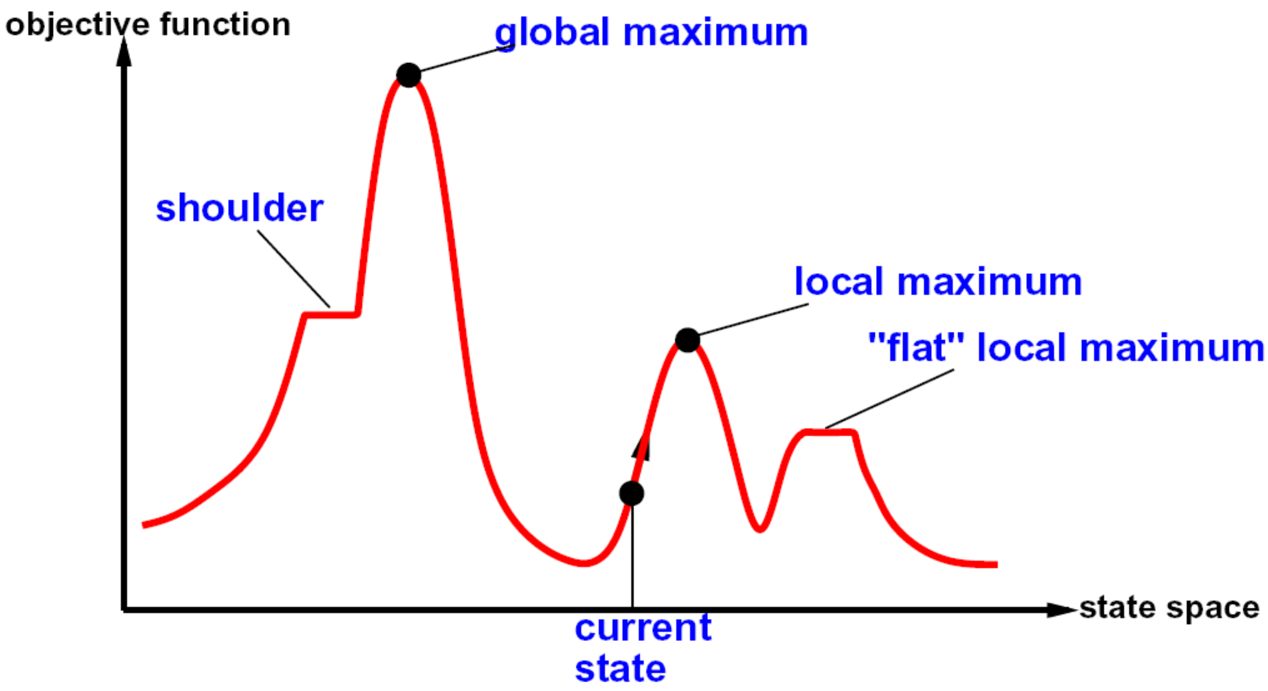


- Generally much faster and more memory efficient (but incomplete and suboptimal)

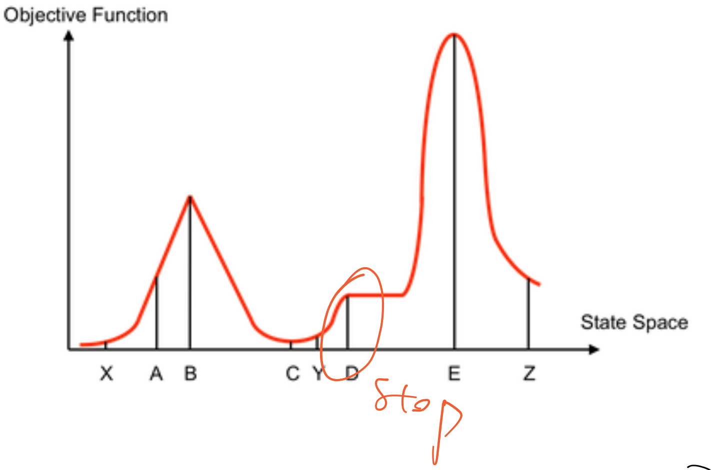
Hill Climbing

- 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



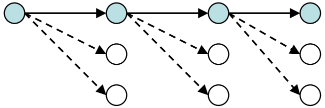
Starting from X, where do you end up ? B

Starting from Y, where do you end up ? D

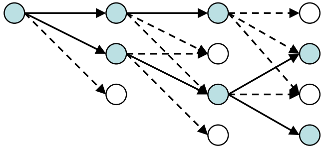
Starting from Z, where do you end up ? E

Beam Search

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



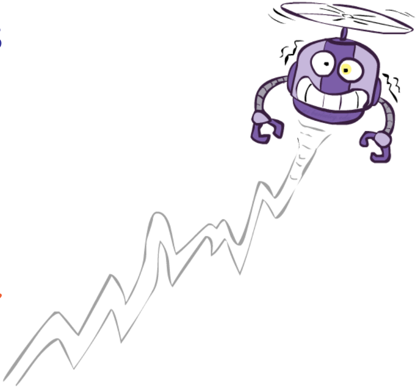
Greedy Search



Beam Search

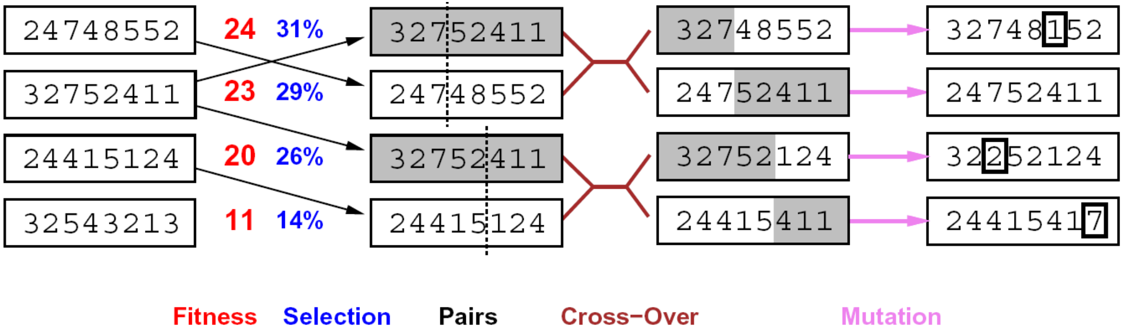
- The best choice in MANY practical settings
- Optimal?

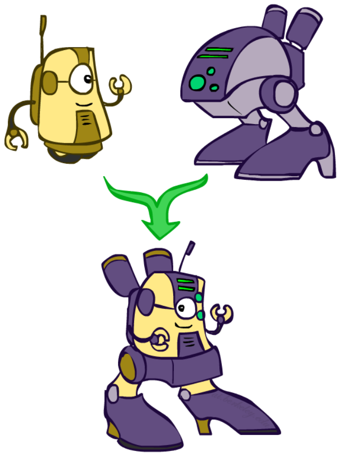
Simulated Annealing



- 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

突变

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