CS 4100 // artificial intelligence

instructor: byron wallace

Recap/midterm review!

Attribution: many of these slides are modified versions of those distributed with the *UC Berkeley CS188* materials
Thanks to John DeNero and Dan Klein

What have we covered?*

- Search!
    - BFS, DFS, UCS
    - A* and informed search
- Constraint Satisfaction Problems (CSPs)
- Adversarial Search / game playing / expecti-max

** Large areas; non-exhaustive*

What have we covered?* (continued)

- Markov Decision Processes (MDPs)
- Reinforcement Learning (RL)
- Markov Models (MMs) / Hidden Markov Models (HMMs)
  - And corresponding probability theory

* *Large areas; non-exhaustive*

What have we covered?*

- **Search!**
  - **BFS, DFS, UCS**
  - **A* and informed search**

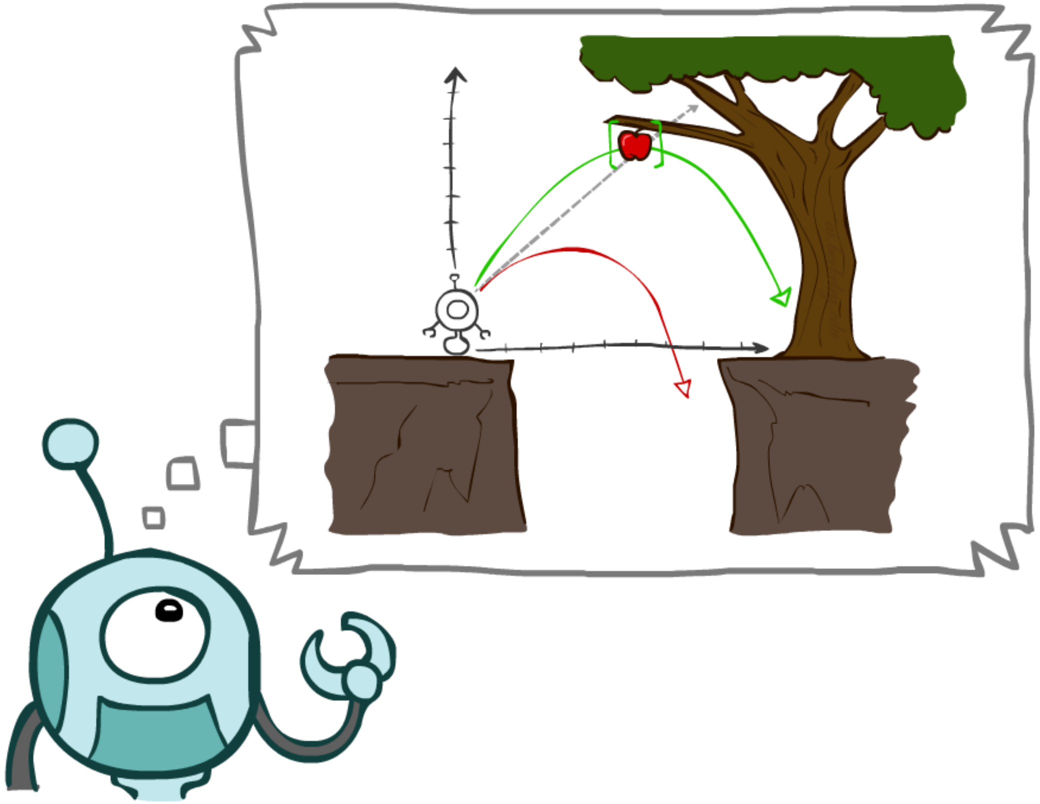
- Constraint Satisfaction Problems (CSPs)

- Adversarial Search / game playing / expecti-max

** Large areas; non-exhaustive*

Search basics

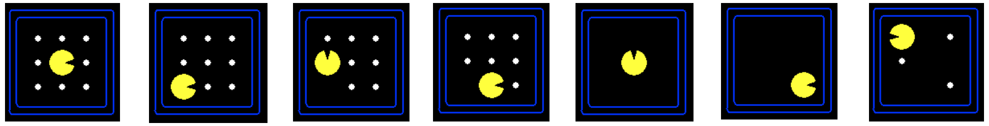
- *Agents* that Plan Ahead
- We will treat plans as *search problems*
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search



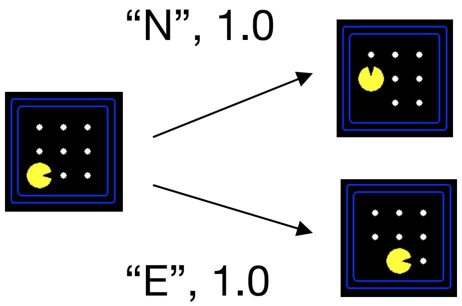
Search problems

A **search problem** consists of:

- A state space



- A successor function (with actions, costs)

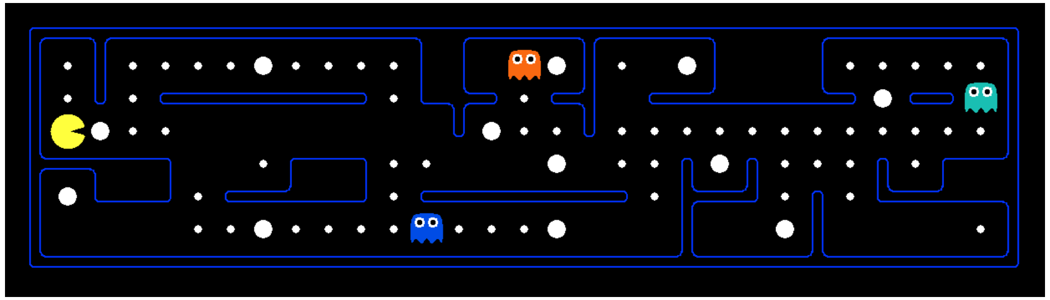


- A start state and a goal test

A **solution** is a sequence of actions (a plan) that transforms the start state to a goal state

World states v. search states

The **world state** includes every last detail of the environment



A **search state** keeps only the details needed for planning (abstraction)

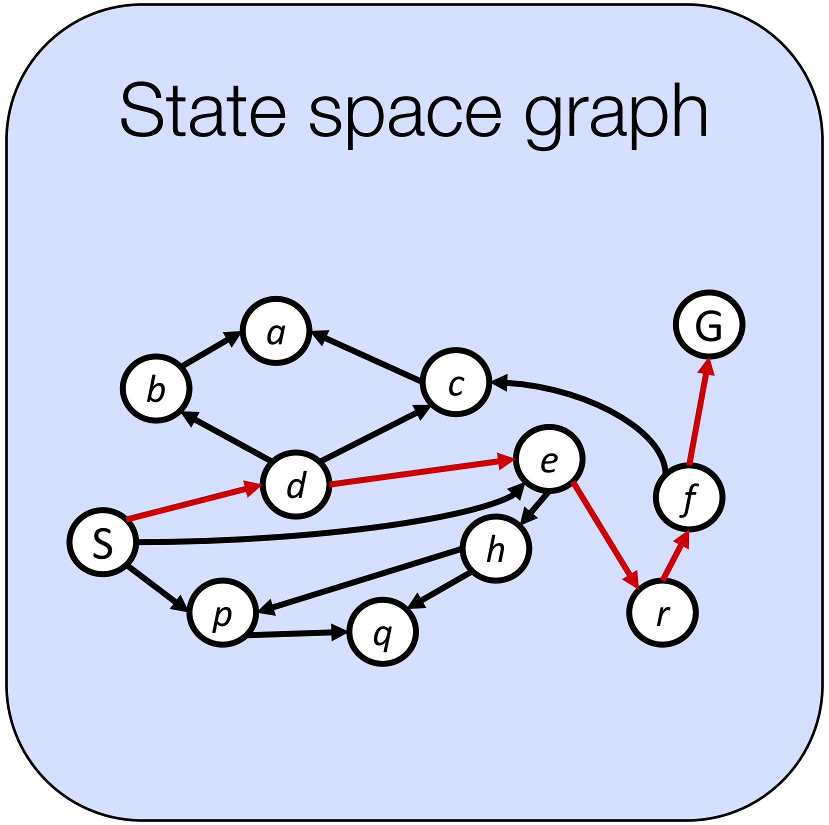
Problem: *Pathing*

- States: (x,y) location
- Actions: NSEW
- Successor: update location only
- Goal test: is (x,y)=END

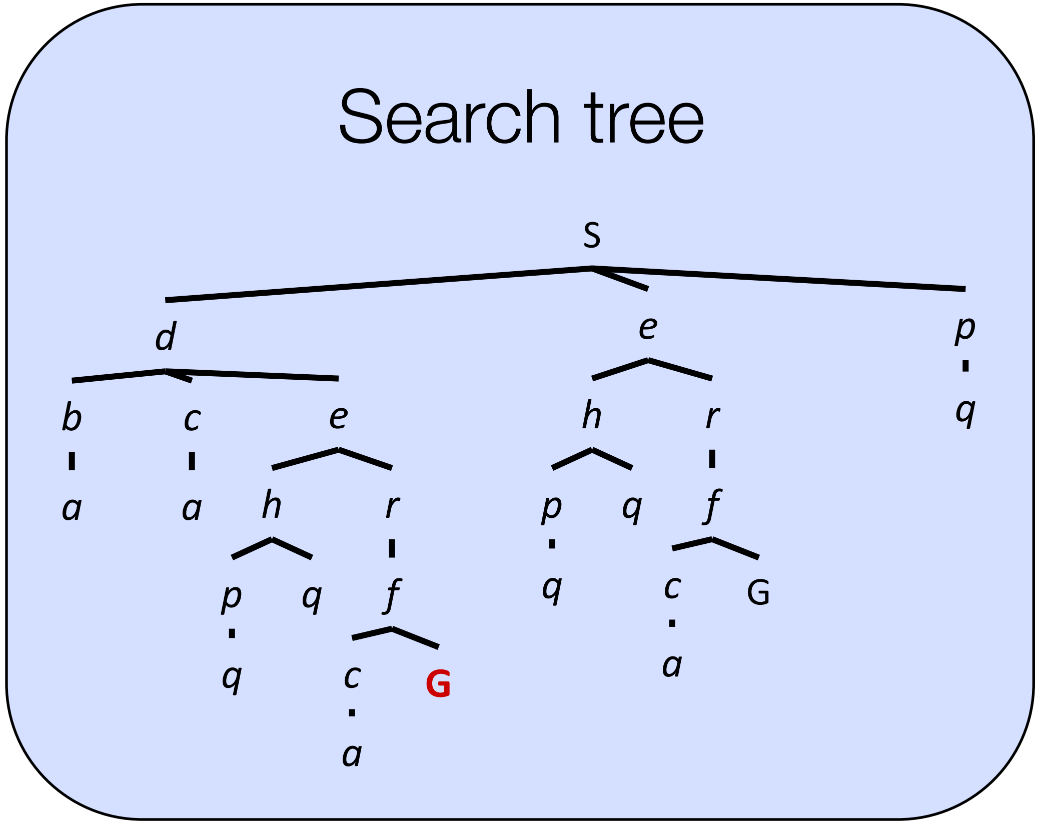
Problem: *Eat-All-Dots*

- States: {(x,y), dot booleans}
- Actions: NSEW
- Successor: update location and possibly a dot boolean (if we eat food)
- Goal test: dots all false

State space graphs vs. search trees



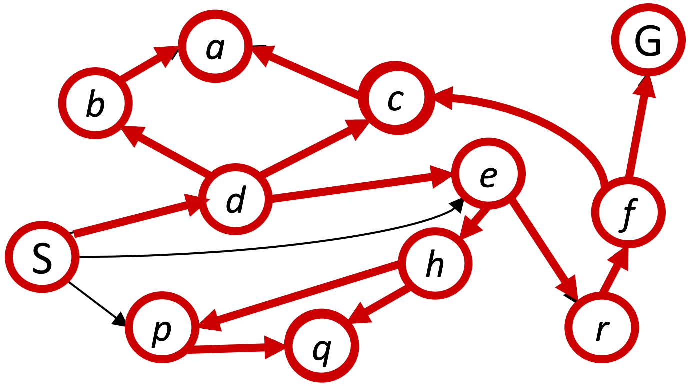
Each *node* in the **search tree** is an *entire path* in the **state space graph**.

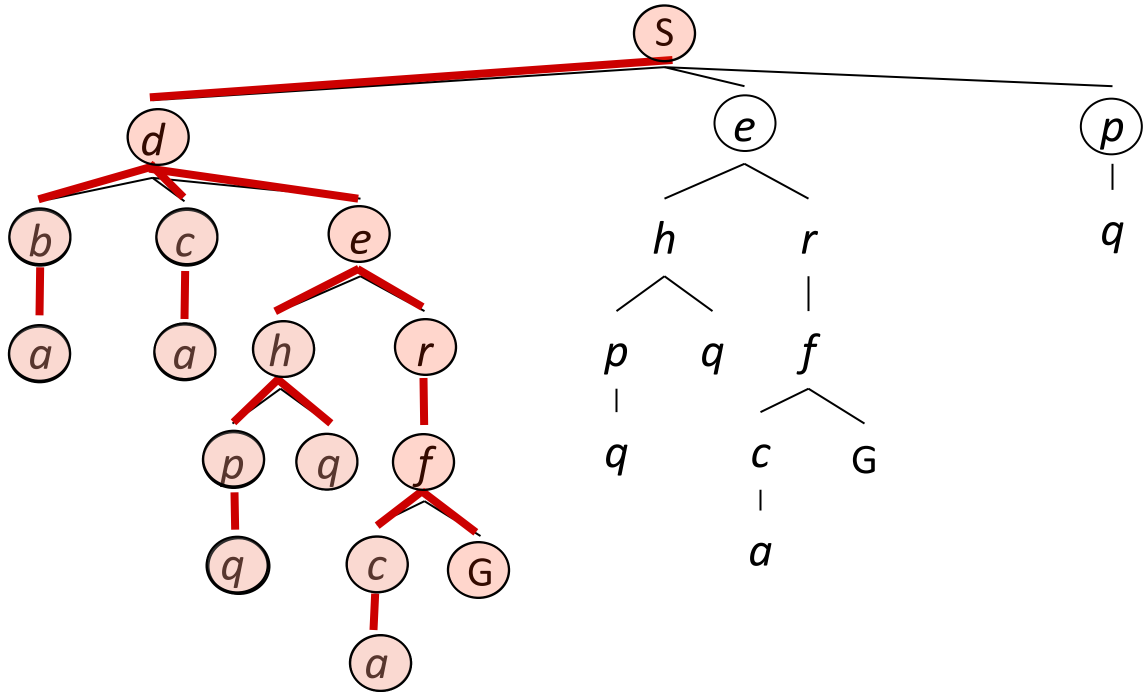


Depth-First Search (DFS)

Strategy expand a deepest node first

Implementation Fringe is a stack (LIFO)

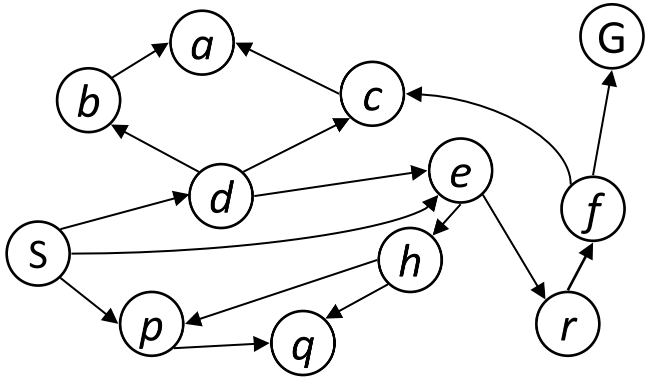




Breadth-First Search (BFS)

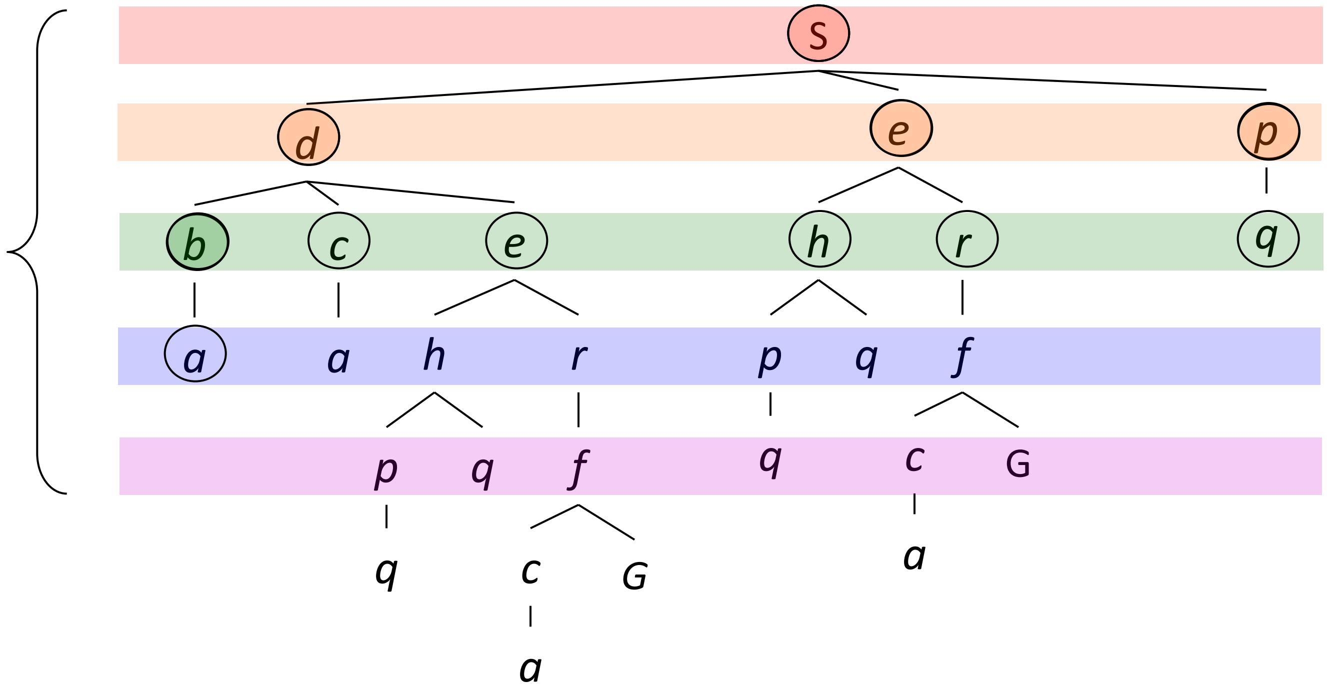
Strategy expand a shallowest node first

Implementation Fringe is a FIFO queue



Search

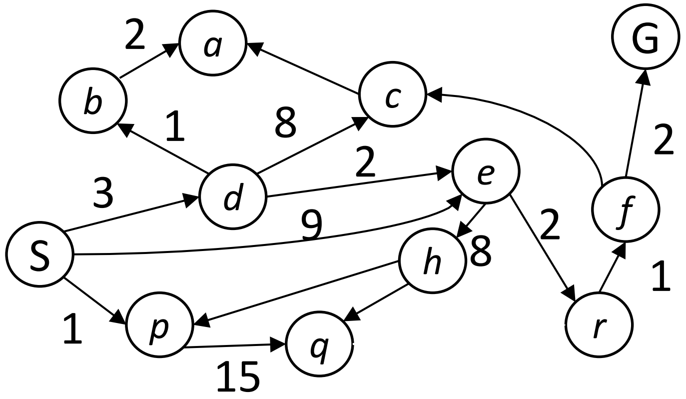
Tiers



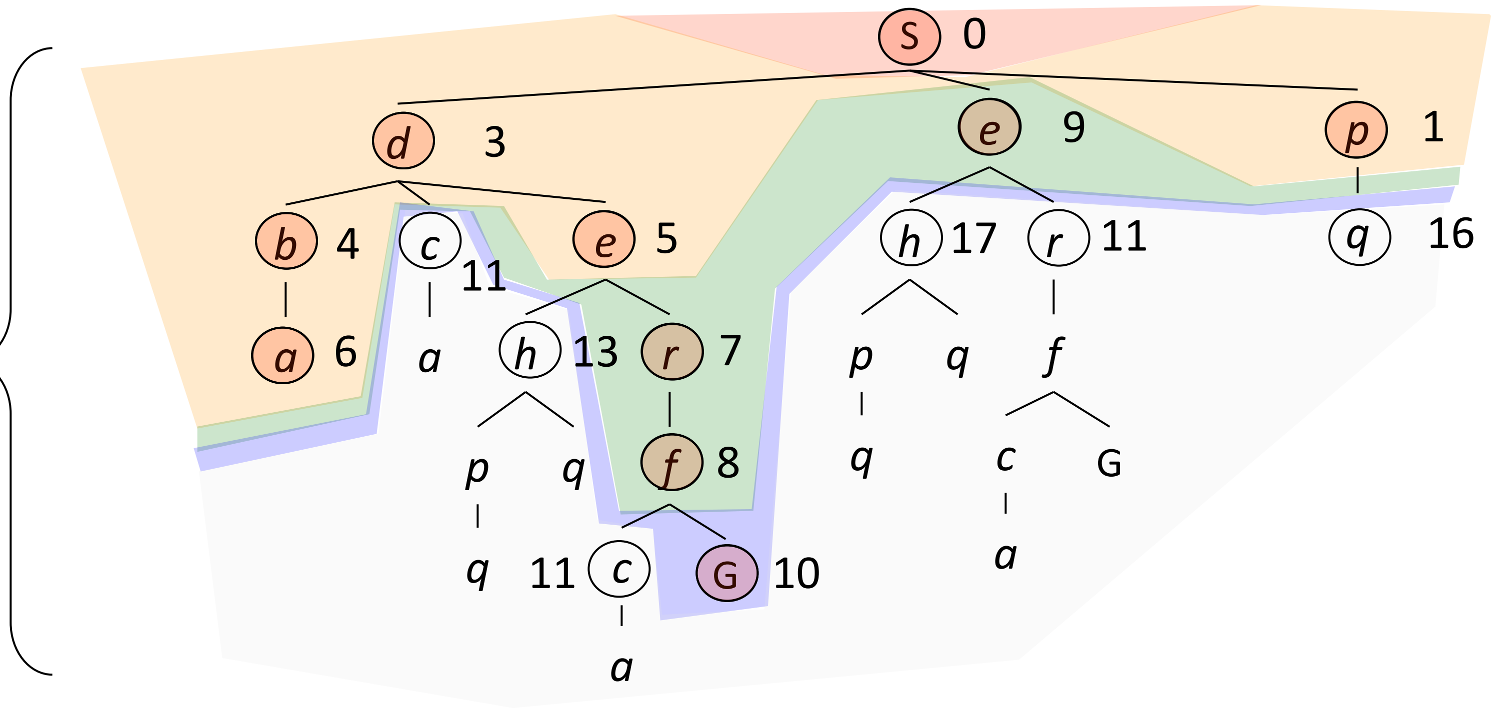
Uniform Cost Search (UCS)

Strategy *expand a cheapest node first:*

Fringe *is a priority queue (priority:* ***cumulative*** *cost)*



Cost contours

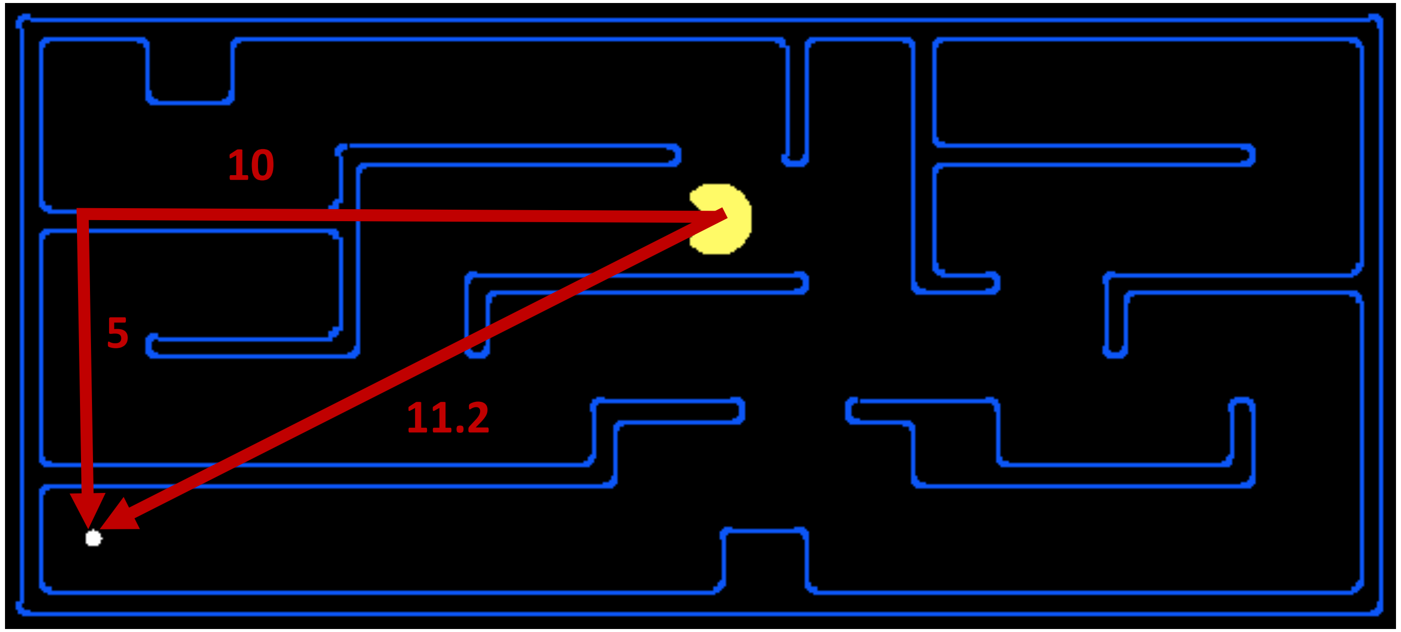


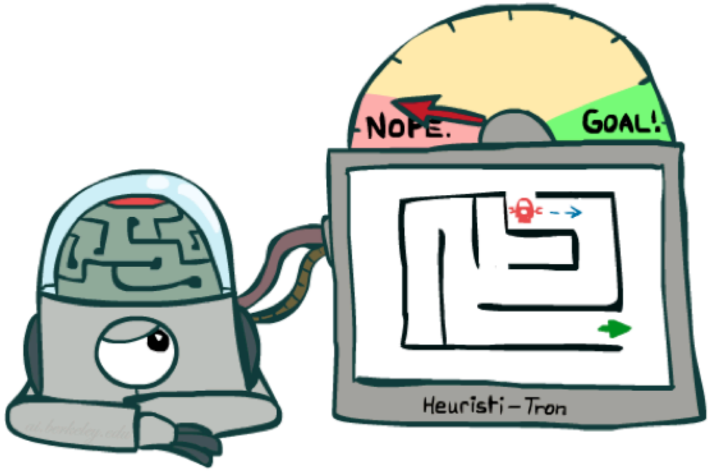
Informed search: A* and beyond

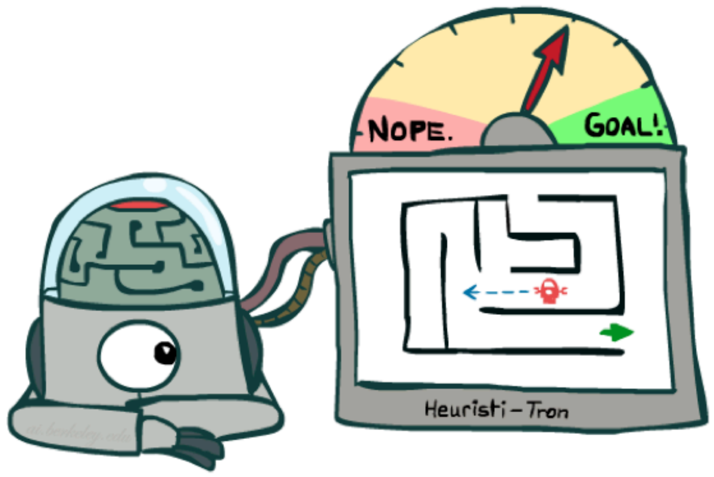
Search heuristics

A *heuristic* is

- A *function* that estimates how close a state is to a goal
- Designed for a particular search problem
- What might we use for PacMan (e.g., for pathing)? Manhattan distance, Euclidean distance

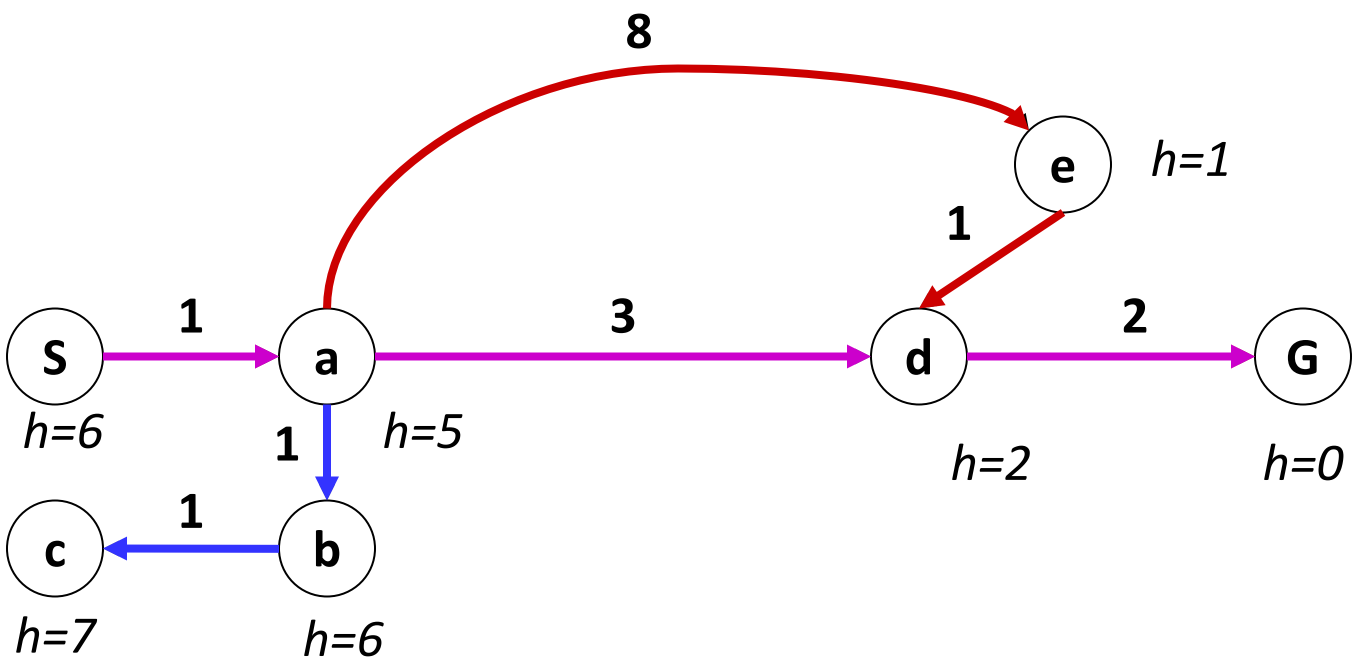


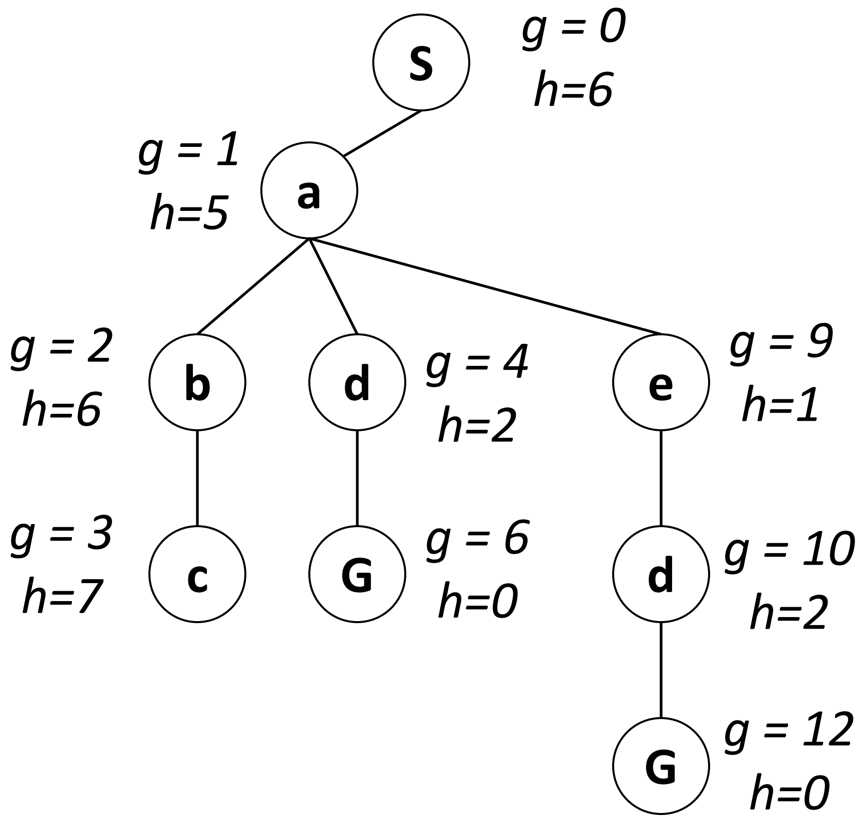




Combining UCS and Greedy

- Uniform-cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)





- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

Admissible heuristics, formally

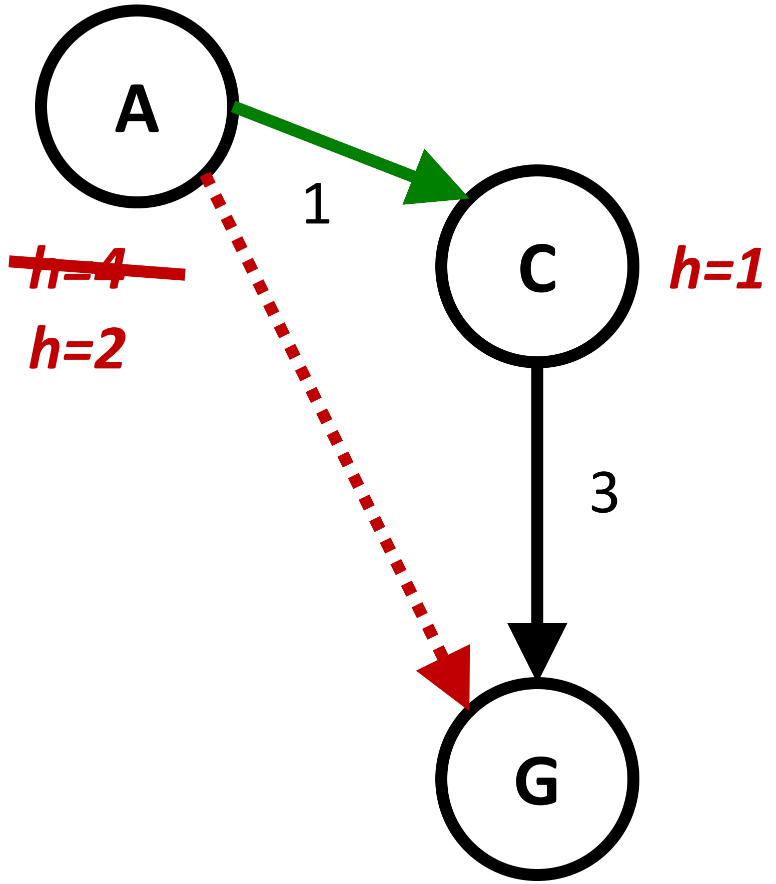
A heuristic h is *admissible* (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal.

Coming up with admissible heuristics is most of what's involved in using A* in practice.

Consistency of heuristics



Main idea: estimated heuristic costs ≤ actual costs

- **Admissibility**: heuristic cost ≤ actual cost to goal

  h(A) ≤ actual cost from A to G

- **Consistency**: heuristic "arc" cost ≤ actual cost for each arc

  h(A) – h(C) ≤ cost(A to C)

Consequences of consistency:

The f value along a path never decreases

h(A) ≤ cost(A to C) + h(C)

A* graph search is optimal

Search example problem

What have we covered?*

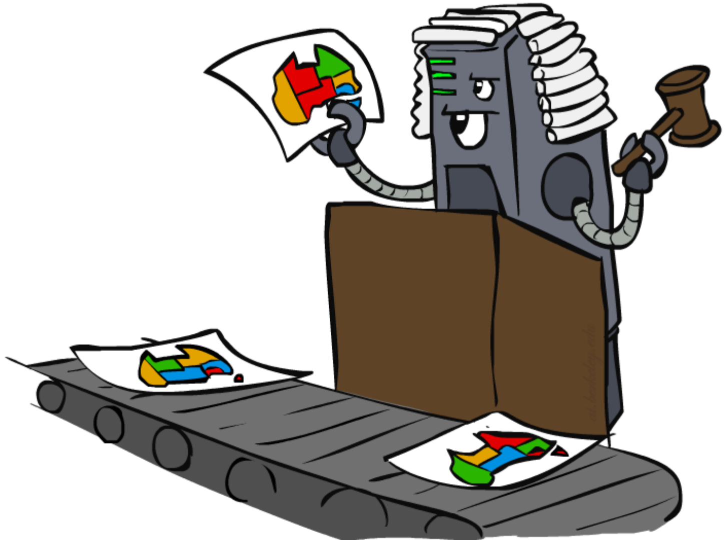
- Search!
  - BFS, DFS, UCS
  - A* and informed search
- **Constraint Satisfaction Problems (CSPs)**
- Adversarial Search / game playing / expecti-max

** Large areas; non-exhaustive*

Constraint Satisfaction Problems (CSPs)

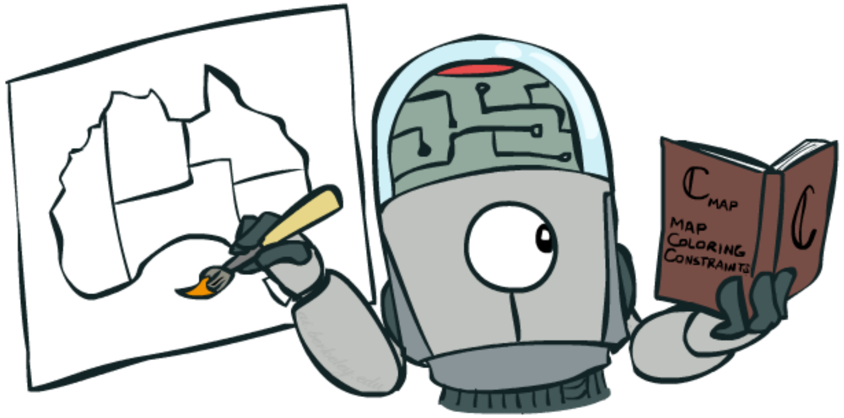
Standard search problems:

- State is a “black box”: arbitrary data structure
- Goal test can be any function over states
- Successor function can also be anything



Constraint satisfaction problems (CSPs):

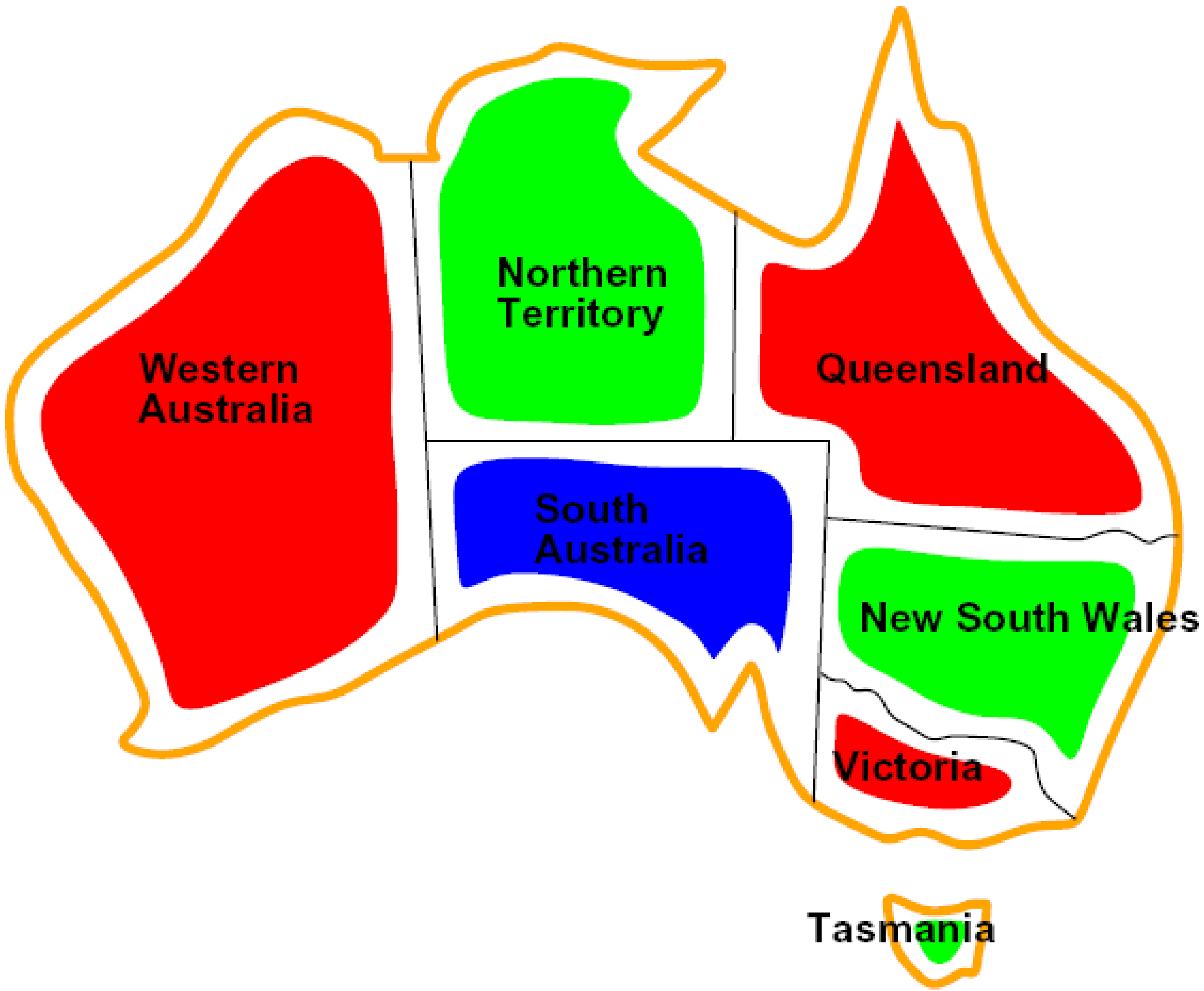
- A special subset of search problems
- State is defined by variables X_i with values from a domain D (sometimes D depends on i)
- *Goal test* is a set of constraints specifying allowable combinations of values for subsets of variables



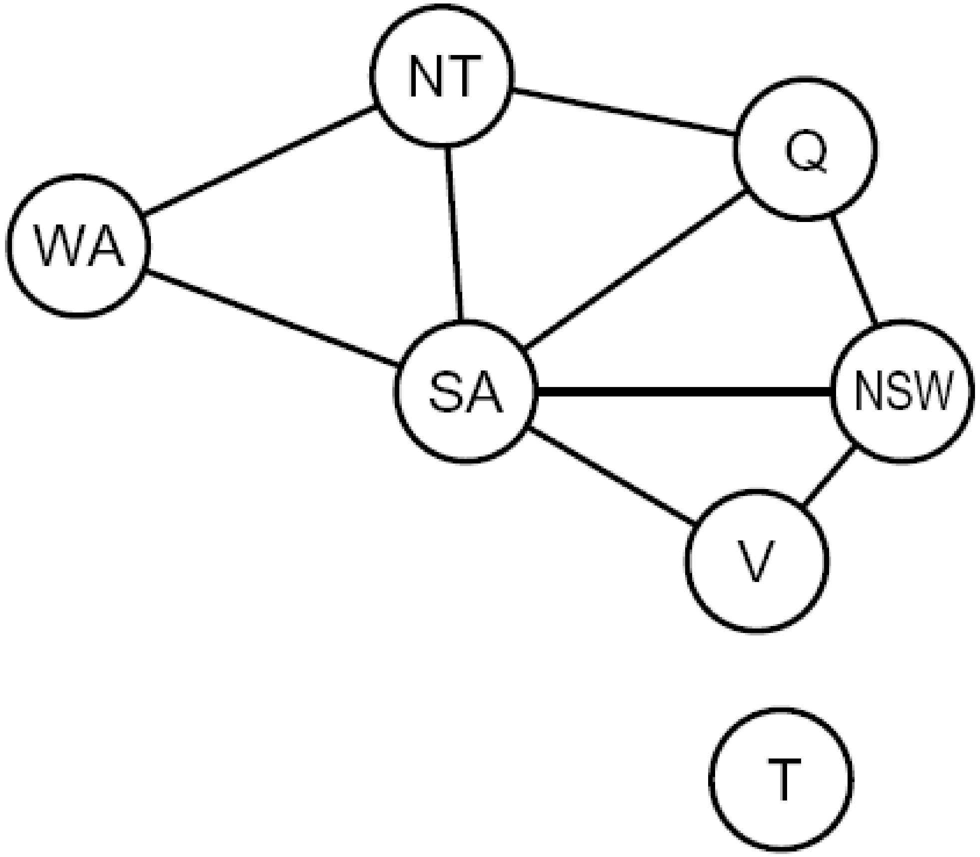
Simple example of a *formal representation language*

Allows useful general-purpose algorithms with more power than standard search algorithms

CSP Examples

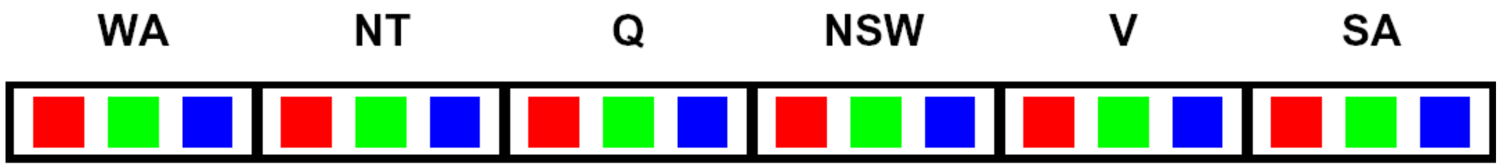


Constraint graphs



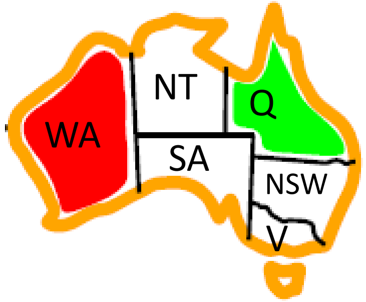
Filtering: forward checking

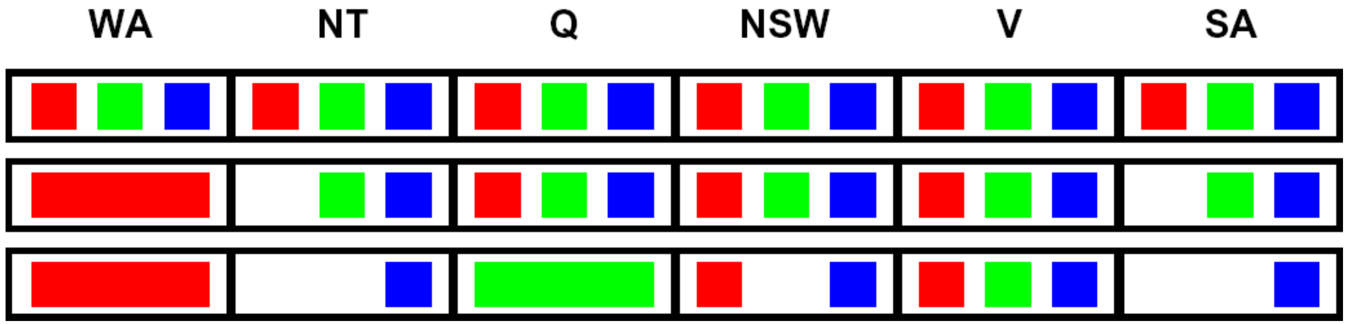
Filtering: Keep track of domains for unassigned variables and cross off bad option



Filtering: constraint propagation

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

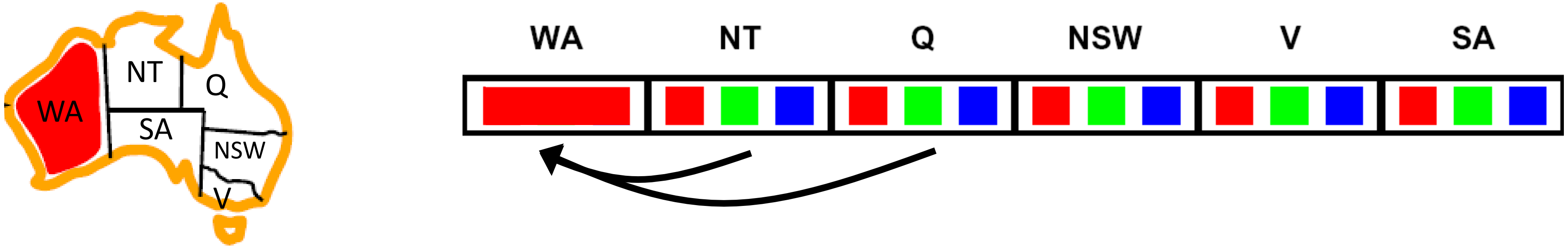




- NT and SA cannot both be blue!
- Why didn’t we detect this yet?
- *Constraint propagation:* reason from constraint to constraint

Consistency of a single arc

An arc X → 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



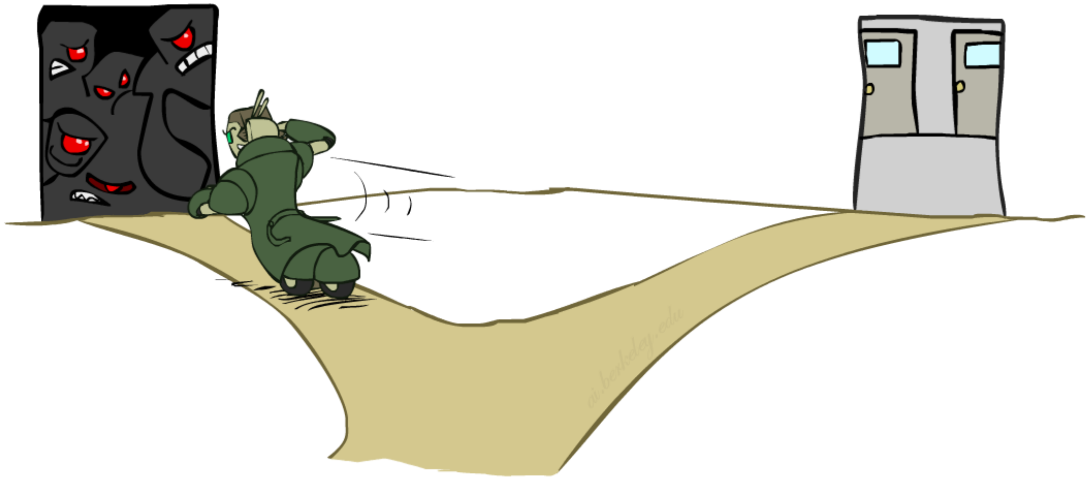
Forward checking: Enforcing consistency of arcs pointing to each new assignment

Ordering: minimum remaining values

Variable Ordering: Minimum remaining values (MRV):

- Choose the variable with the fewest legal left values in its domain

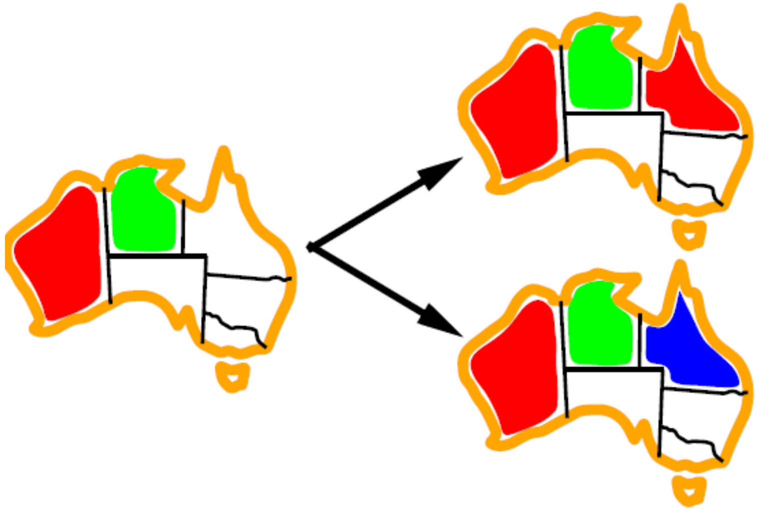
- Why min rather than max?
- Also called “most constrained variable”
- “Fail-fast” ordering



Ordering: least constraining value

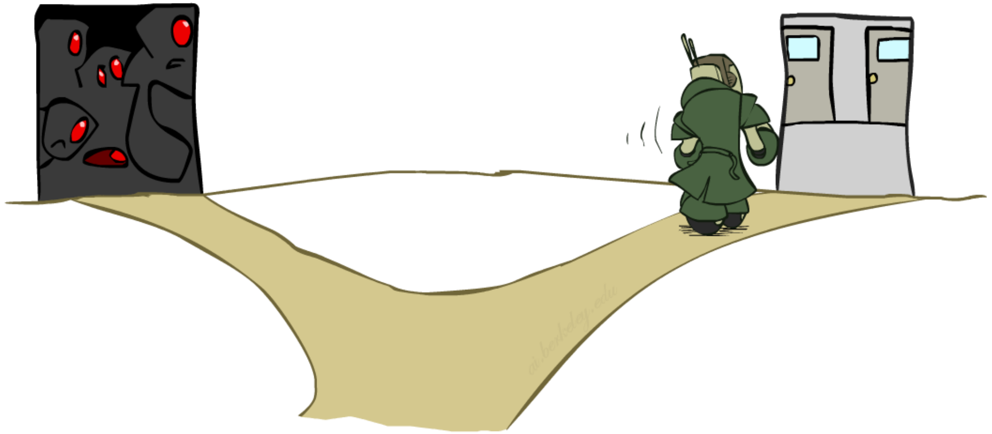
Value Ordering: *Least Constraining Value*

- 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?

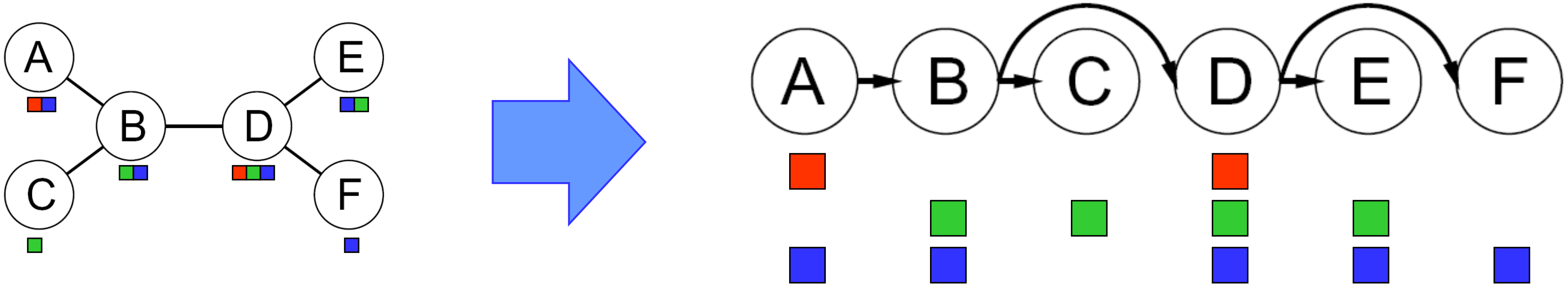
Combining these ordering ideas makes 1000 queens feasible



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)

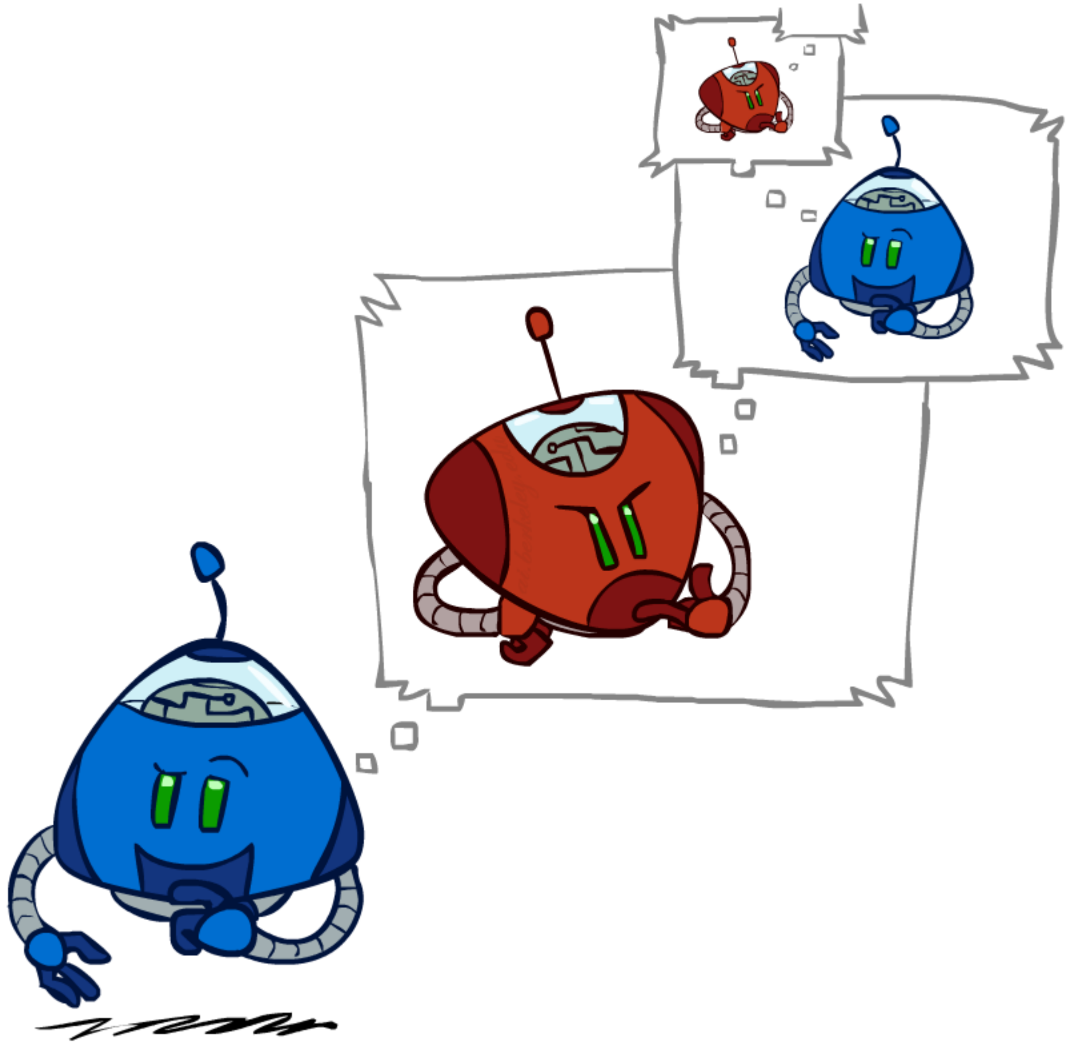
CSP example problem

What have we covered?*

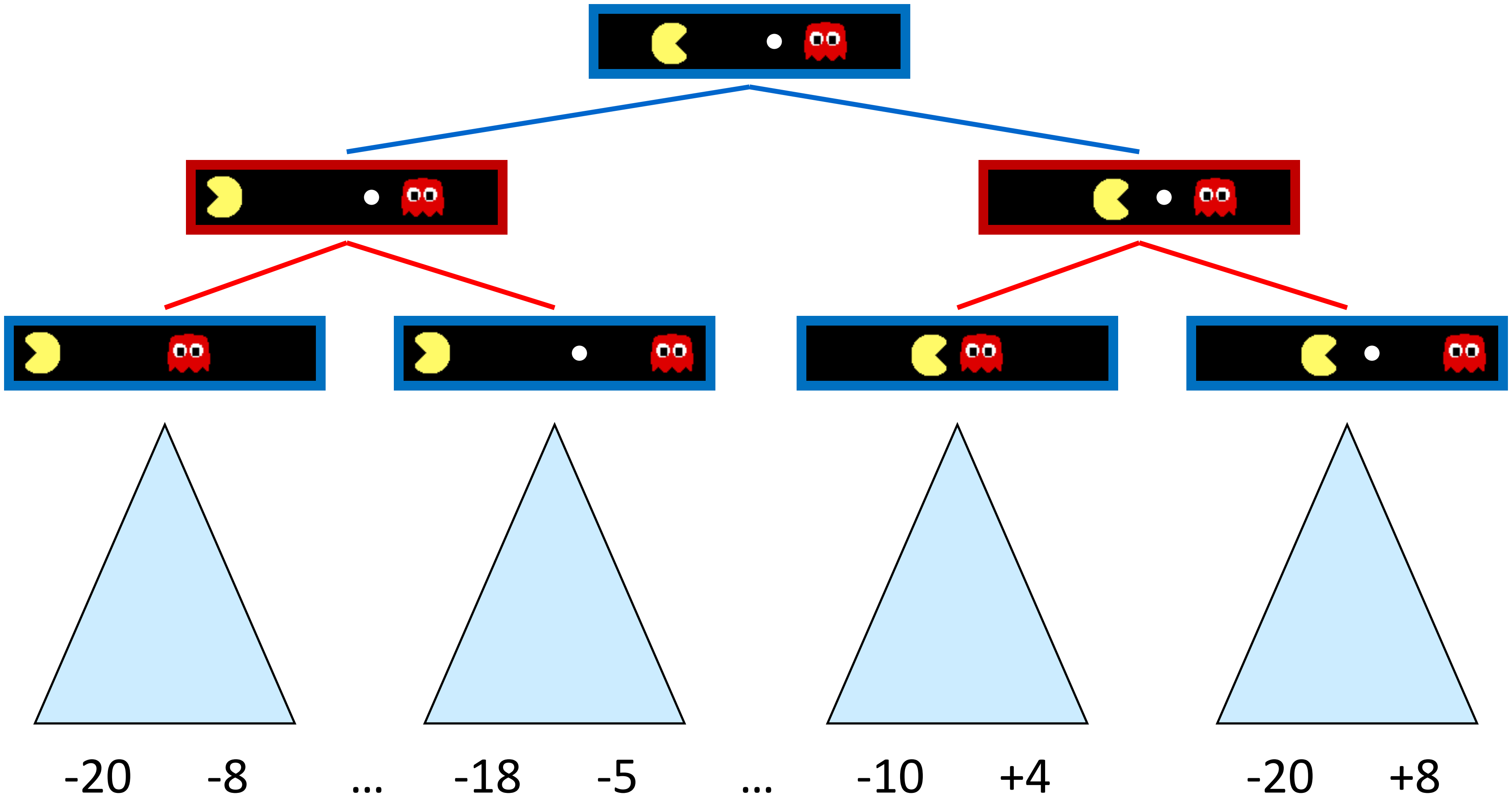
- Search!
    - BFS, DFS, UCS
    - A* and informed search
- Constraint Satisfaction Problems (CSPs)
- **Adversarial Search / game playing / expecti-max**

** Large areas; non-exhaustive*

Adversarial search



Adversarial game trees



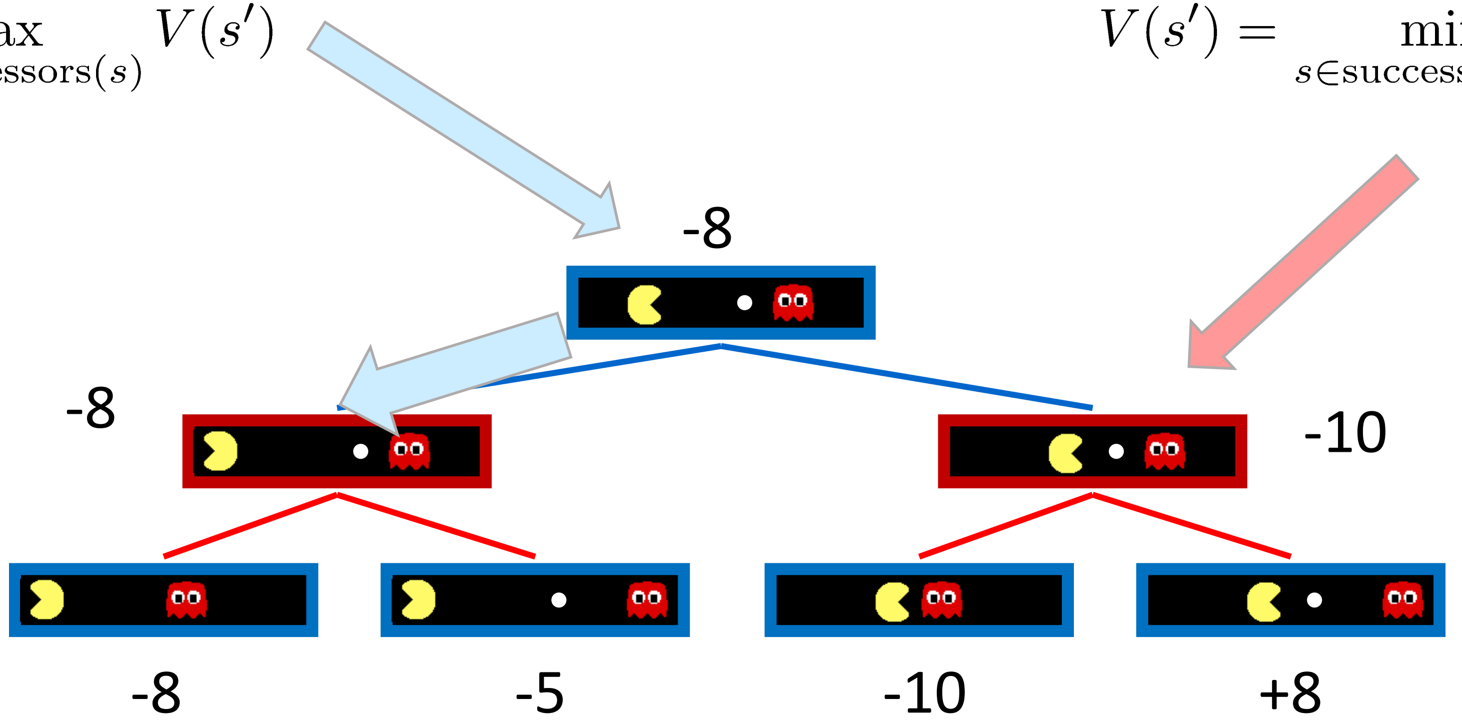
Minimax values

States Under Agent's Control:

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

States Under Opponent's Control:

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

$$V(s) = \text{known}$$

Adversarial search (Minimax)

Deterministic, zero-sum games:

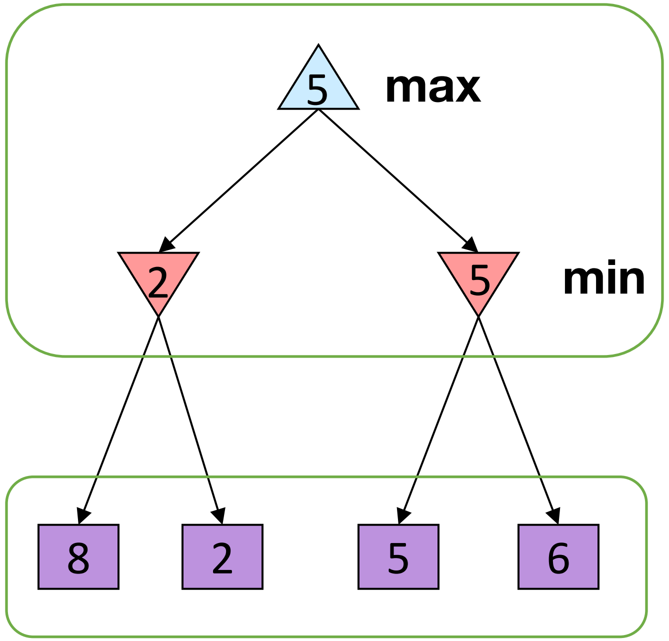
- Tic-tac-toe, chess, checkers
- One player maximizes result
- The other minimizes result

Minimax search:

- A state-space search tree
- Players alternate turns
- Compute each node's **minimax value:** the best achievable utility against a rational (optimal) adversary

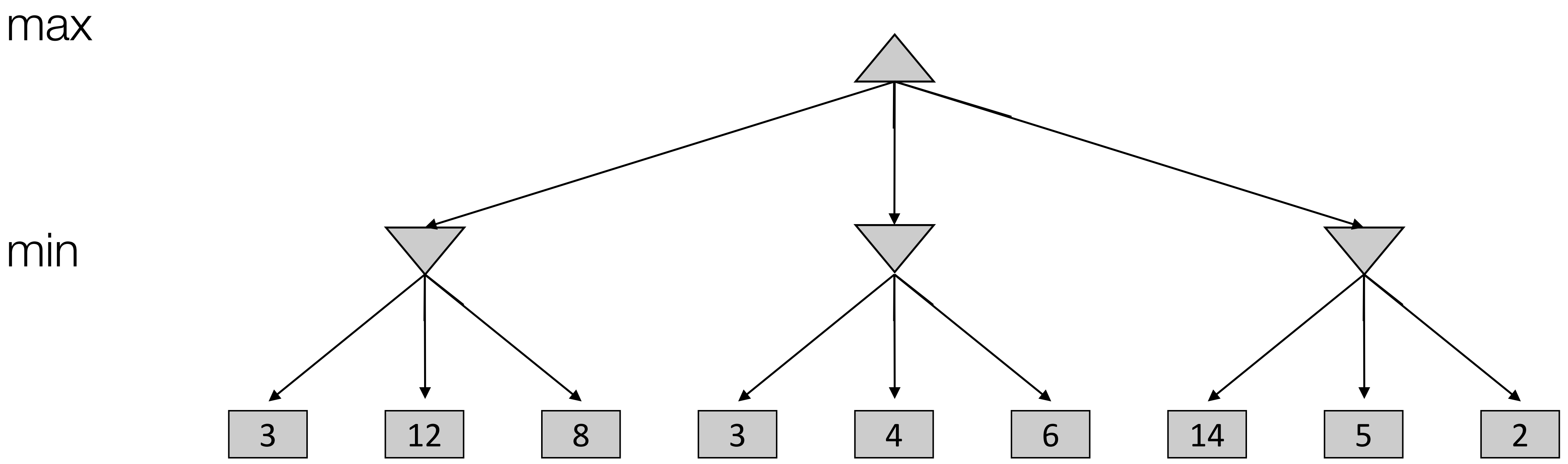
$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

Minimax values: computed recursively



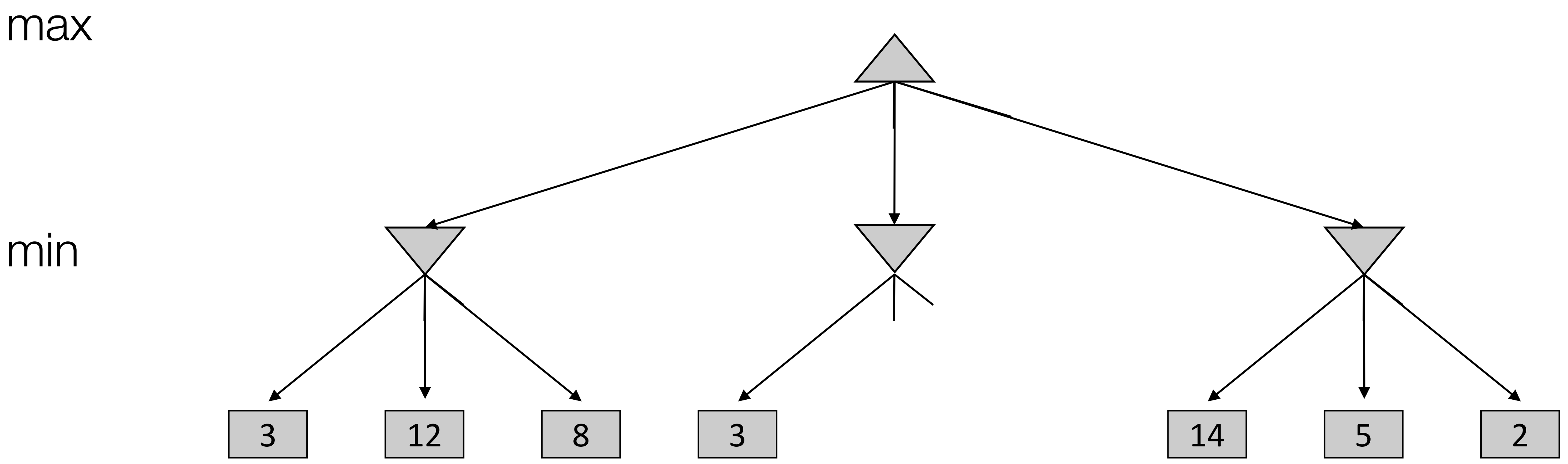
Terminal values: part of the game

Pruning



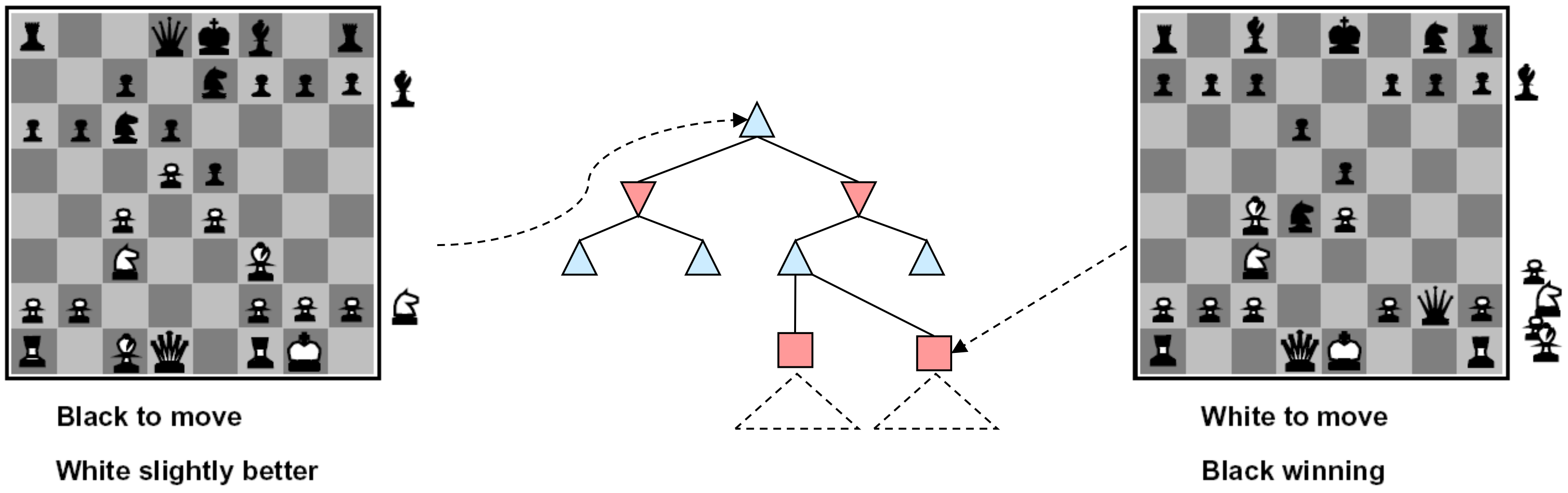
What did we do that was inefficient here?

Minimax pruning



Evaluation functions

Evaluation functions score non-terminals in depth-limited search



Ideal function: returns the actual minimax value of the position
In practice: typically weighted linear sum of features:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g. $f_1(s)$ = (num white queens – num black queens), etc.

Adversarial search example problem

What have we covered?* (continued)

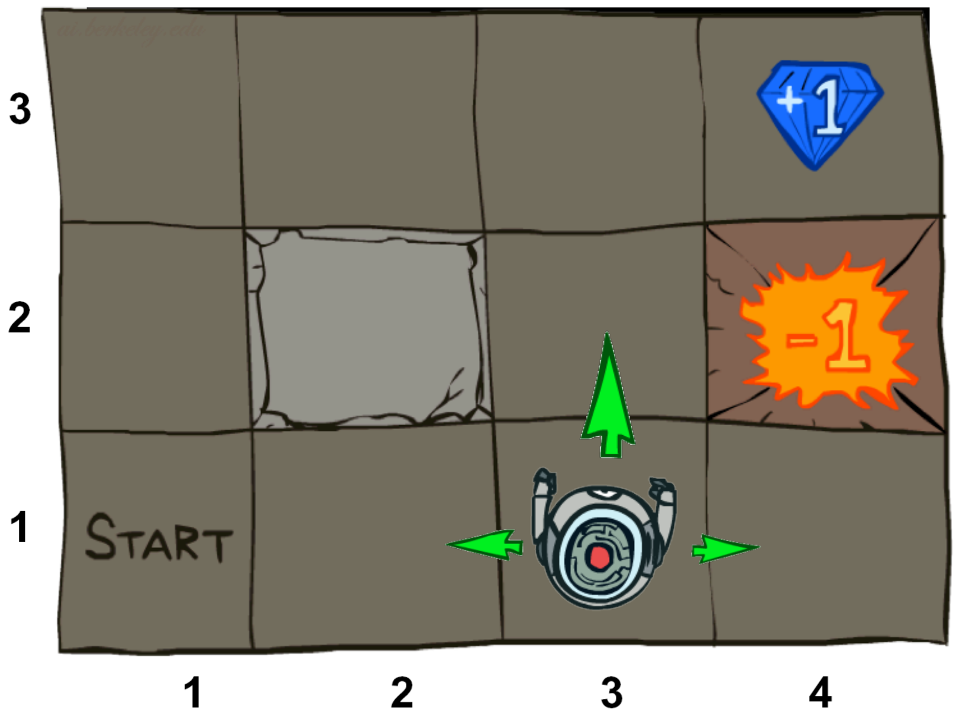
- **Markov Decision Processes (MDPs)**
- Reinforcement Learning (RL)
- Markov Models (MMs) / Hidden Markov Models (HMMs)
  - And corresponding probability theory

** Large areas; non-exhaustive*

Markov Decision Processes

An MDP is defined by

- States $\in$ S
- Actions a $\in$ A
- Transition function T(s, a, s')
  Probability that a from s leads to s', i.e., P(s'| s, a)
  Also called the model or the dynamics
- Reward function R(s, a, s')
  Sometimes just R(s) or R(s')
- Start state
- Maybe a terminal state



MDPs are *non-deterministic* search problems

- One way to solve them is with *expectimax* search; but we'll do better
- They can go on *forever*
- Arguably, life is an MDP

Optimal quantities

The value (utility) of a state s

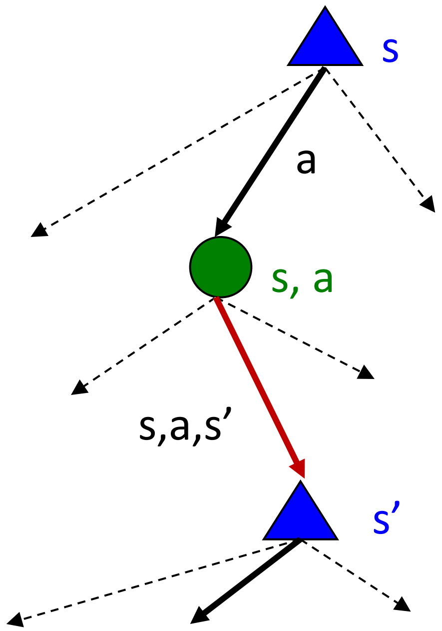
$V^*(s)$ = *expected utility* starting in s and acting optimally

The value (utility) of a q-state (s,a)

$Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

The optimal policy

$\pi^*(s)$ = optimal action from state s



s is a *state*

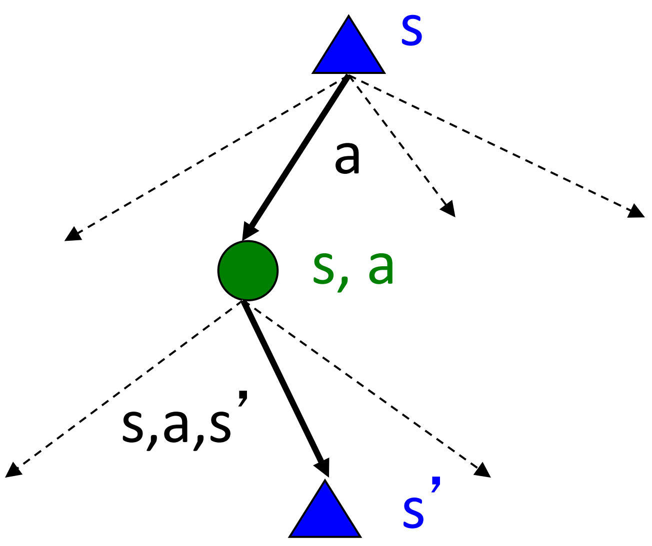
(s, a) is a *q-state*

(s,a,s') is a *transition*

Values of states

Fundamental operation: compute the (expectimax) value of a state

- Expected utility under optimal action
- This will be an average sum of (discounted) rewards
- This is just what expectimax computed!



Recursive definition of value:

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

Value iteration

- Start at bottom with $V_0(s) = 0$: no time steps left means an expected reward sum of zero

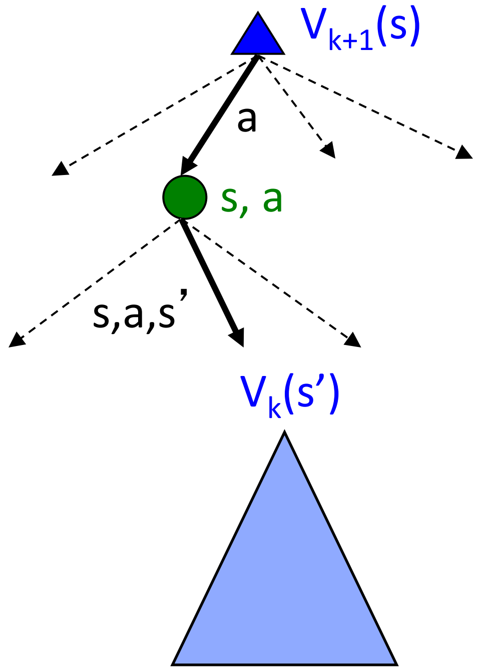
- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

- Repeat until convergence

- Complexity of each iteration: $O(S^2A)$

- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values (but will only converge if we use a discount / have a finite horizon!)
  - Policy often converges before values!



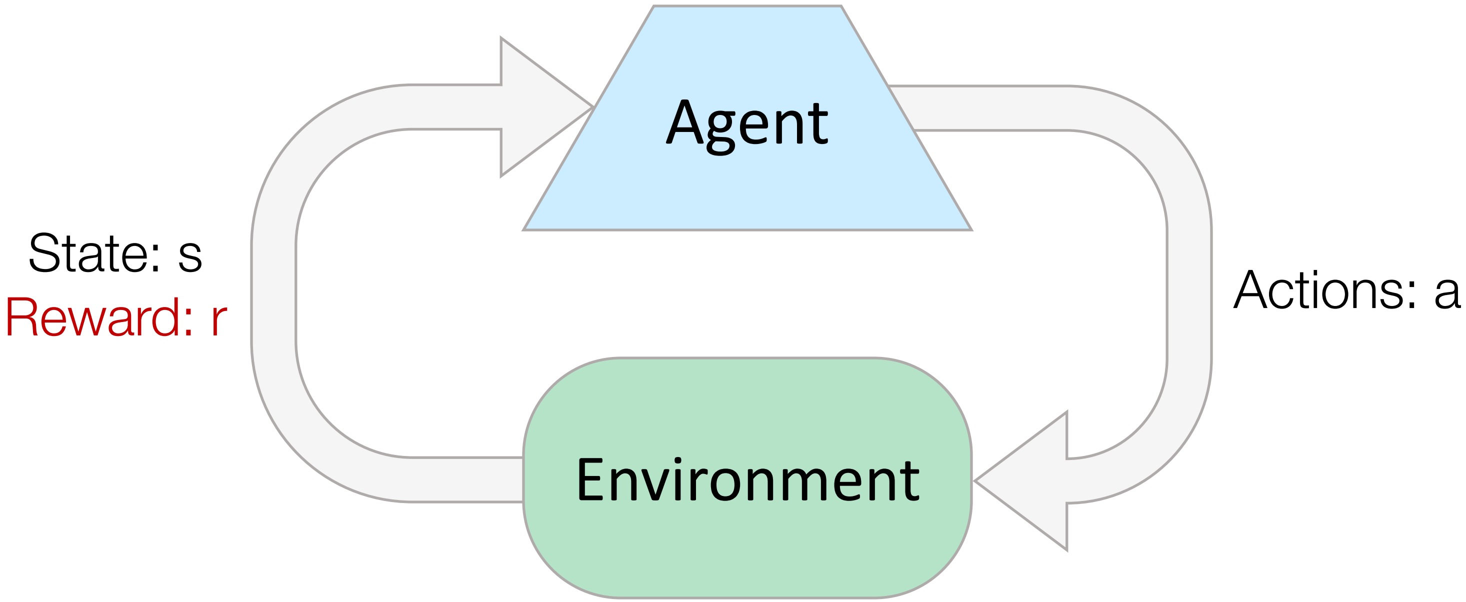
MDPs problem

What have we covered?* (continued)

- Markov Decision Processes (MDPs)
- **Reinforcement Learning (RL)**
- Markov Models (MMs) / Hidden Markov Models (HMMs)
    - And corresponding probability theory

** Large areas; non-exhaustive*

Reinforcement learning



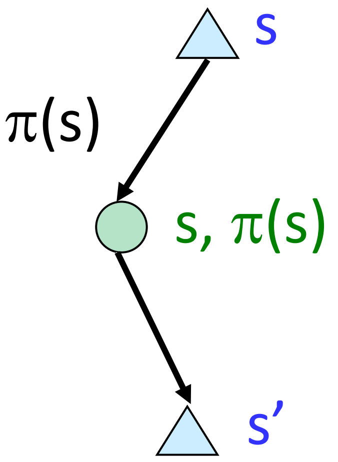
Basic idea:

- Receive feedback in the form of **rewards**
- Agent's utility is defined by the reward function
- Must (learn to) act so as to **maximize expected rewards**
- All learning is based on observed samples of outcomes!

Temporal Difference Learning (model free!)

Big idea: learn from every experience!

- Update V(s) each time we experience a transition (s, a, s’, r)
- Likely outcomes s’ will contribute updates more often



Temporal difference learning of values

- Policy still fixed, **still doing evaluation**!
- Move values toward value of whatever successor occurs: running average

Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha) sample$

Can rewrite as: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

Q-Learning

Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

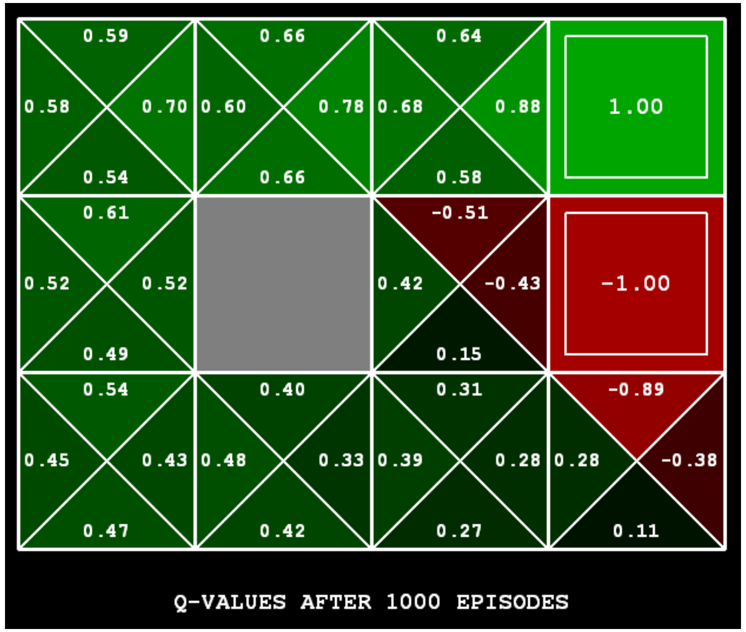
Learn Q(s,a) values as you go

- Receive a sample (s,a,s',r)
- Consider your old estimate: $Q(s,a)$
- Consider your new sample estimate:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

- Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$



Exploration functions

When to explore?

- Random actions: explore a fixed amount
- Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

Exploration function

- Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g.

$$f(u, n) = u + k/n$$

Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

Modified Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

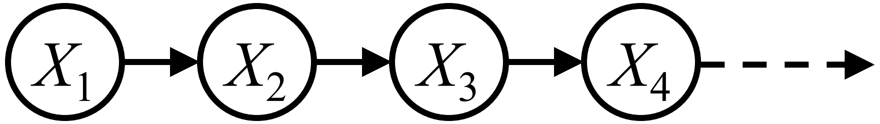
RL problem example

What have we covered?* (continued)

- Markov Decision Processes (MDPs)
- Reinforcement Learning (RL)
- **Markov Models (MMs) / Hidden Markov Models (HMMs)**
  - **And corresponding probability theory**

** Large areas; non-exhaustive*

Chain rule and Markov models



From the chain rule, every joint distribution over $X_1, X_2, \ldots, X_T$ can be written as:

$$P(X_1, X_2, \ldots, X_T) = P(X_1) \prod_{t=2}^{T} P(X_t | X_1, X_2, \ldots, X_{t-1})$$

Assuming that for all t:

$$X_t \perp\!\!\!\perp X_1, \ldots, X_{t-2} \mid X_{t-1}$$

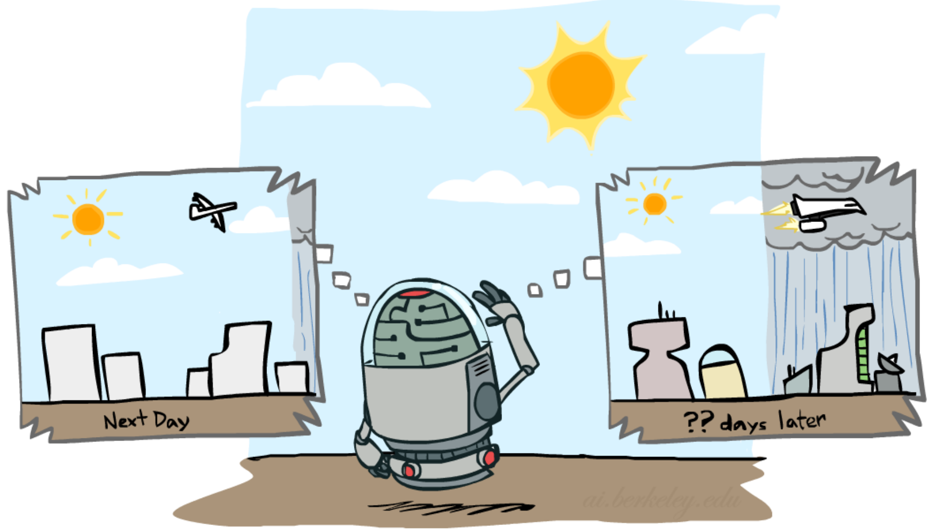
Gives us the expression posited on the earlier slide:

$$P(X_1, X_2, \ldots, X_T) = P(X_1) \prod_{t=2}^{T} P(X_t | X_{t-1})$$

Mini-forward algorithm

Question: What's P(X) on some day t?

$$X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4 \dashrightarrow$$



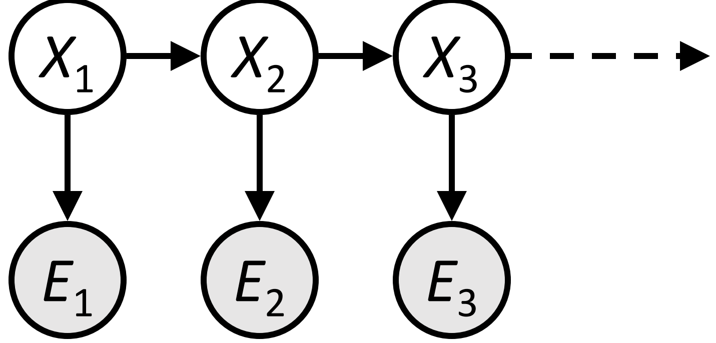
$$P(x_1) = \text{known}$$

$$P(x_t) = \sum_{x_{t-1}} P(x_{t-1}, x_t)$$

$$= \sum_{x_{t-1}} P(x_t \mid x_{t-1}) P(x_{t-1})$$

Forward simulation

The chain rule and HMMs, in general



From the chain rule, every joint distribution over $X_1, E_1, \ldots, X_T, E_T$ can be written as:

$$P(X_1, E_1, \ldots, X_T, E_T) = P(X_1)P(E_1|X_1)\prod_{t=2}^{T} P(X_t|X_1, E_1, \ldots, X_{t-1}, E_{t-1})P(E_t|X_1, E_1, \ldots, X_{t-1}, E_{t-1}, X_t)$$

Assuming that for all *t*:

State independent of all past states and all past evidence given the previous state, i.e.:

$$X_t \perp\!\!\!\perp X_1, E_1, \ldots, X_{t-2}, E_{t-2}, E_{t-1} \mid X_{t-1}$$

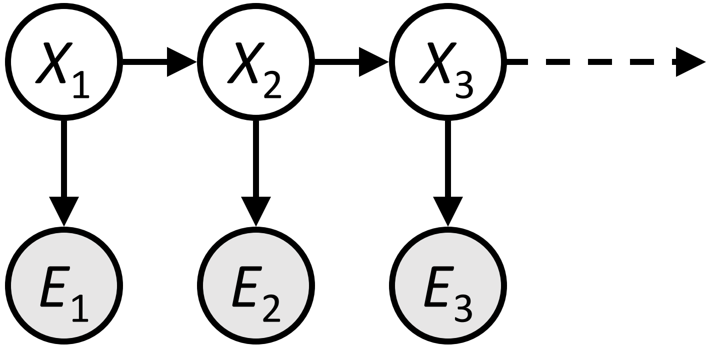
Evidence is independent of all past states and all past evidence given the current state, i.e.:

$$E_t \perp\!\!\!\perp X_1, E_1, \ldots, X_{t-2}, E_{t-2}, X_{t-1}, E_{t-1} \mid X_t$$

Which gives us:

$$P(X_1, E_1, \ldots, X_T, E_T) = P(X_1)P(E_1|X_1)\prod_{t=2}^{T} P(X_t|X_{t-1})P(E_t|X_t)$$

Implied conditional independencies



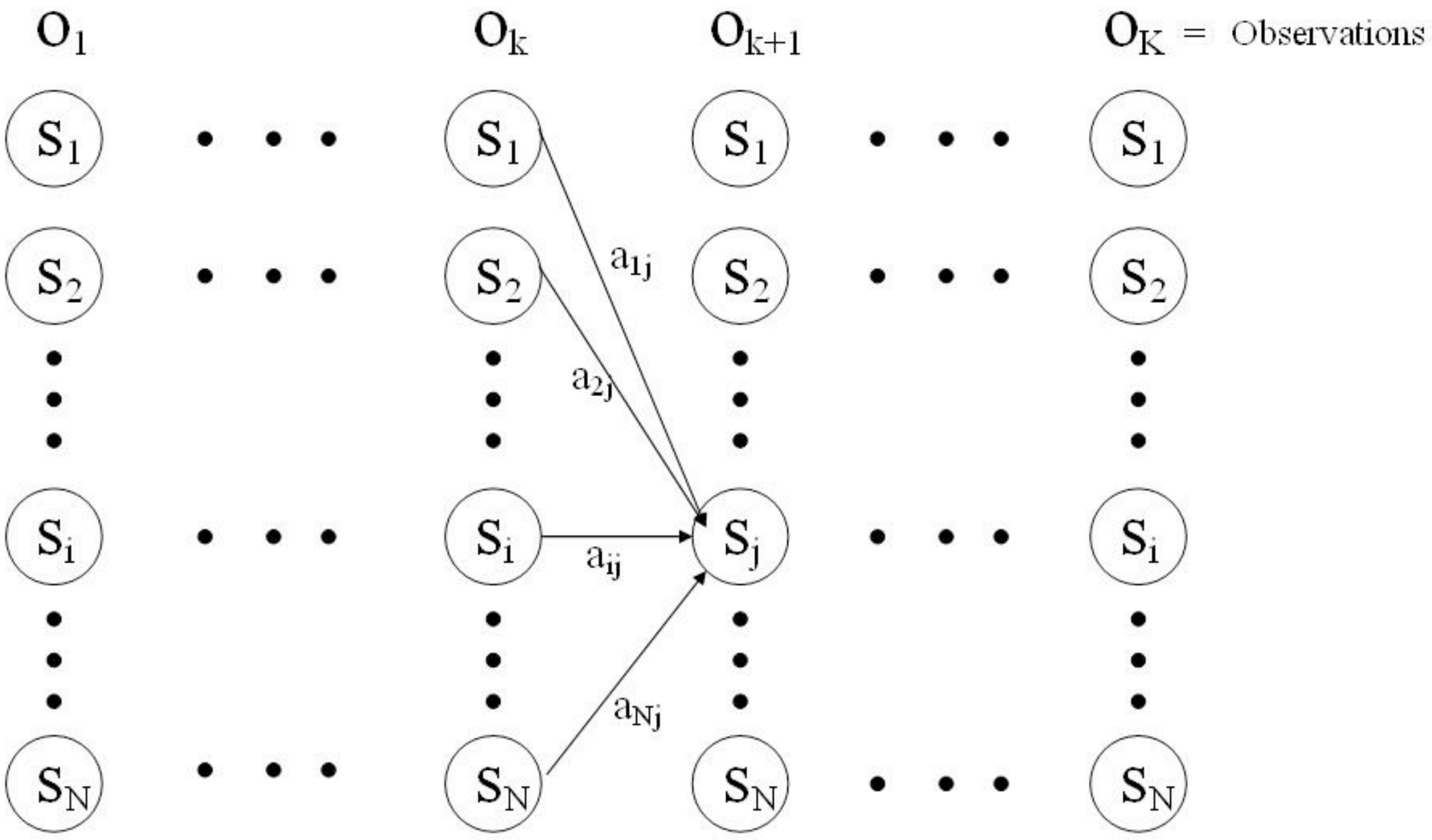
Many implied conditional independencies, e.g.,

$$E_1 \perp\!\!\!\perp X_2, E_2, X_3, E_3 \mid X_1$$

We can prove these as we did last class for Markov models (but we won't today)

This also comes from the graphical model; we'll cover this more formally in a later lecture

Dynamic programming (the “forward algorithm”)



time T

Markov model problem example