CS 343H: Honors AI

Lecture 10: MDPs I

2/18/2014

Kristen Grauman

UT Austin

Slides courtesy of Dan Klein, UC Berkeley

Unless otherwise noted

Some context

- First weeks: search (BFS, A*, minimax, alpha beta)
  - Find an optimal plan (or solution)
  - Best thing to do from the current state
  - Assume we know transition function and cost (reward) function
  - Either execute complete solution (deterministic) or search again at every step
- Last week: detour for probabilities and utilities
- This week: MDPs – towards reinforcement learning
  - Still know transition and reward function
  - Looking for a policy – optimal action from every state
- Next week: reinforcement learning
  - Optimal policy without knowing transition or reward function

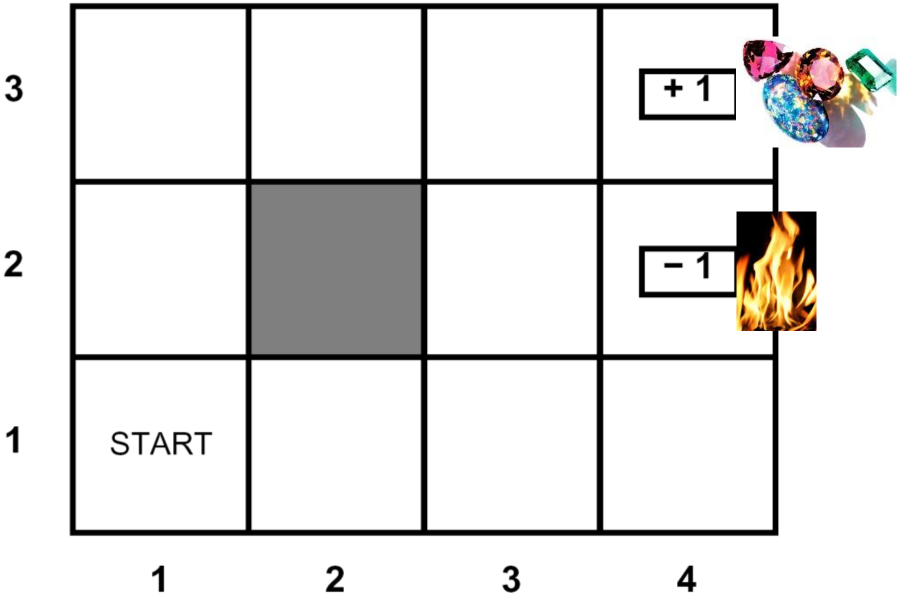
Slide credit: Peter Stone

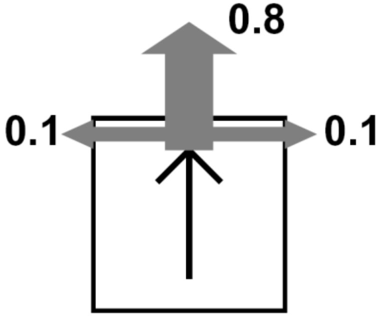
Non-Deterministic Search

How do you plan when your actions might fail?

Example: Grid World

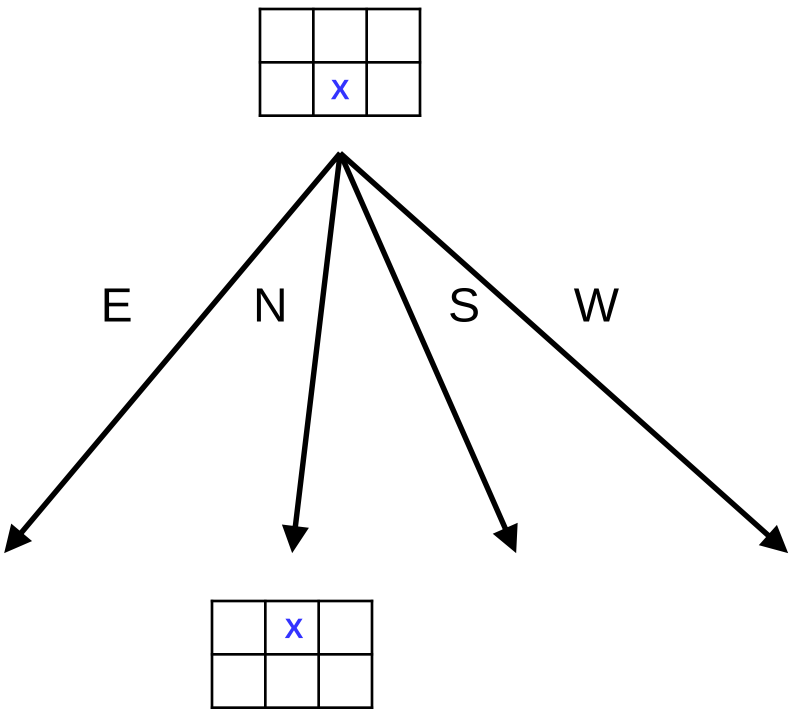
- The agent lives in a grid
- Walls block the agent's path
- The agent's actions do not always go as planned:
  - 80% of the time, the action North takes the agent North
(if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Small "living" reward each step
- Big rewards come at the end
- Goal: maximize sum of rewards



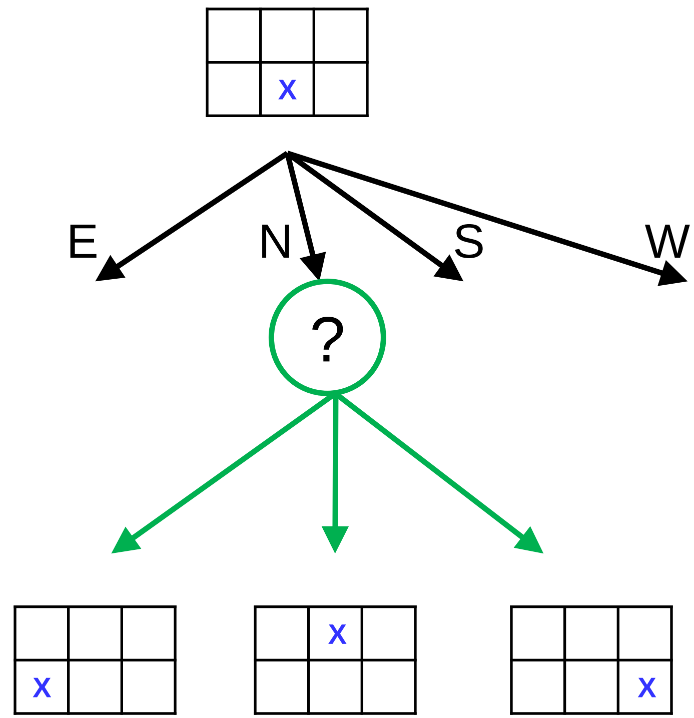


Action Results

Deterministic Grid World

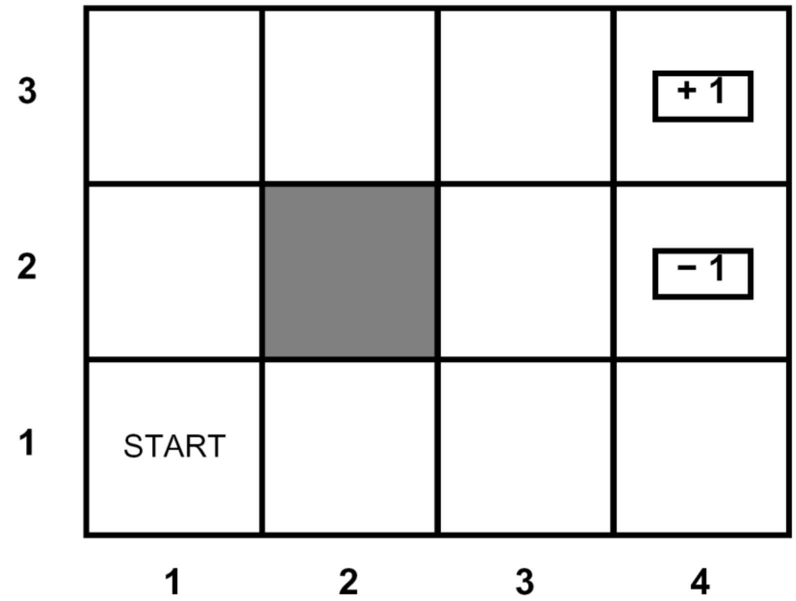


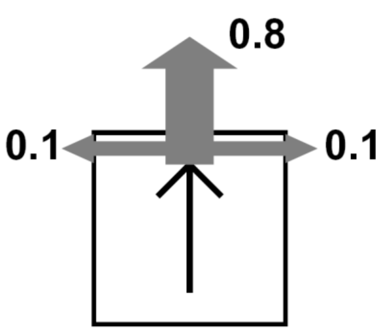
Stochastic Grid World



Markov Decision Processes

- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ A
  - A transition function T(s,a,s')
    - Prob that a from s leads to s'
    - i.e., P(s' | s,a)
    - Also called the model
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state (or distribution)
  - Maybe a terminal state





- MDPs are a family of non-deterministic search problems
  - One way to solve them is with expectimax search – but we'll have a new tool soon

What is Markov about MDPs?

- "Markov" generally means that given the present state, the future and the past are independent

- For Markov decision processes, "Markov" means action outcomes depend only on the current state:

Andrey Markov (1856-1922)

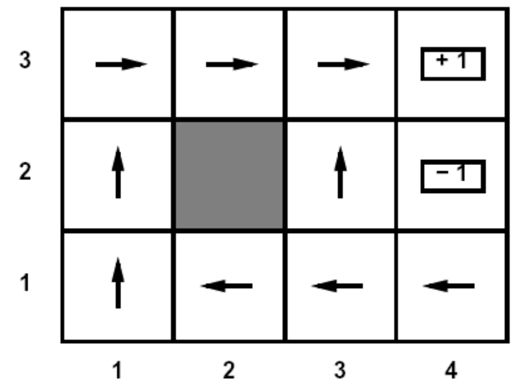
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \dots S_0 = s_0)$$

$$=$$

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

Solving MDPs: Policies

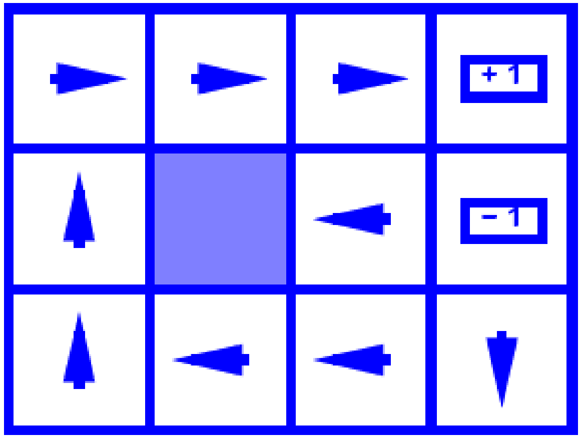
- In deterministic single-agent search problems, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy maximizes expected utility if followed
  - Defines a reflex agent (if precomputed)



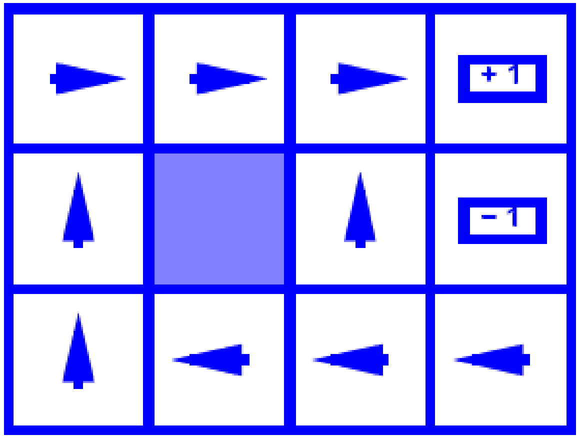
Optimal policy when $R(s, a, s') = -0.03$ for all non-terminals s

- Expectimax didn't compute entire policies
  - It computed the action for a single state only

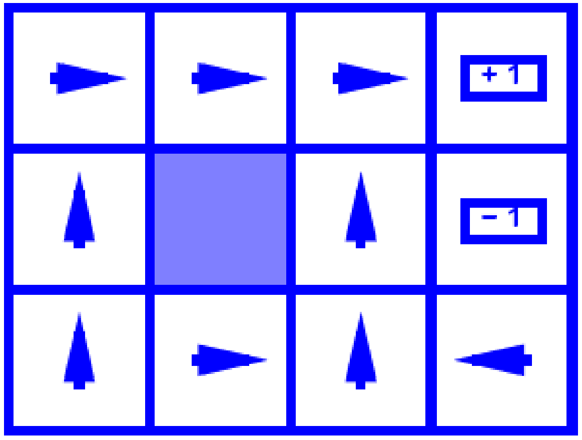
Optimal Policies



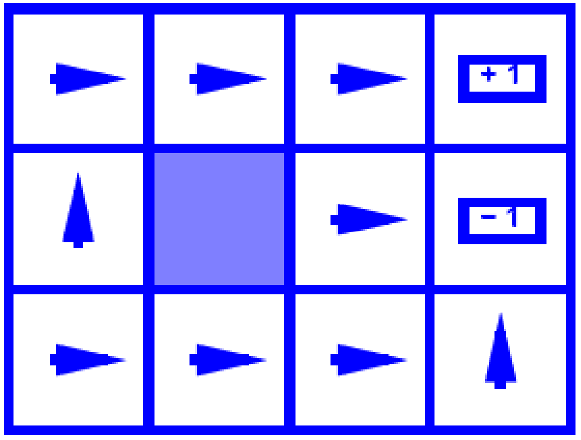
R(s) = -0.01



R(s) = -0.03



R(s) = -0.4

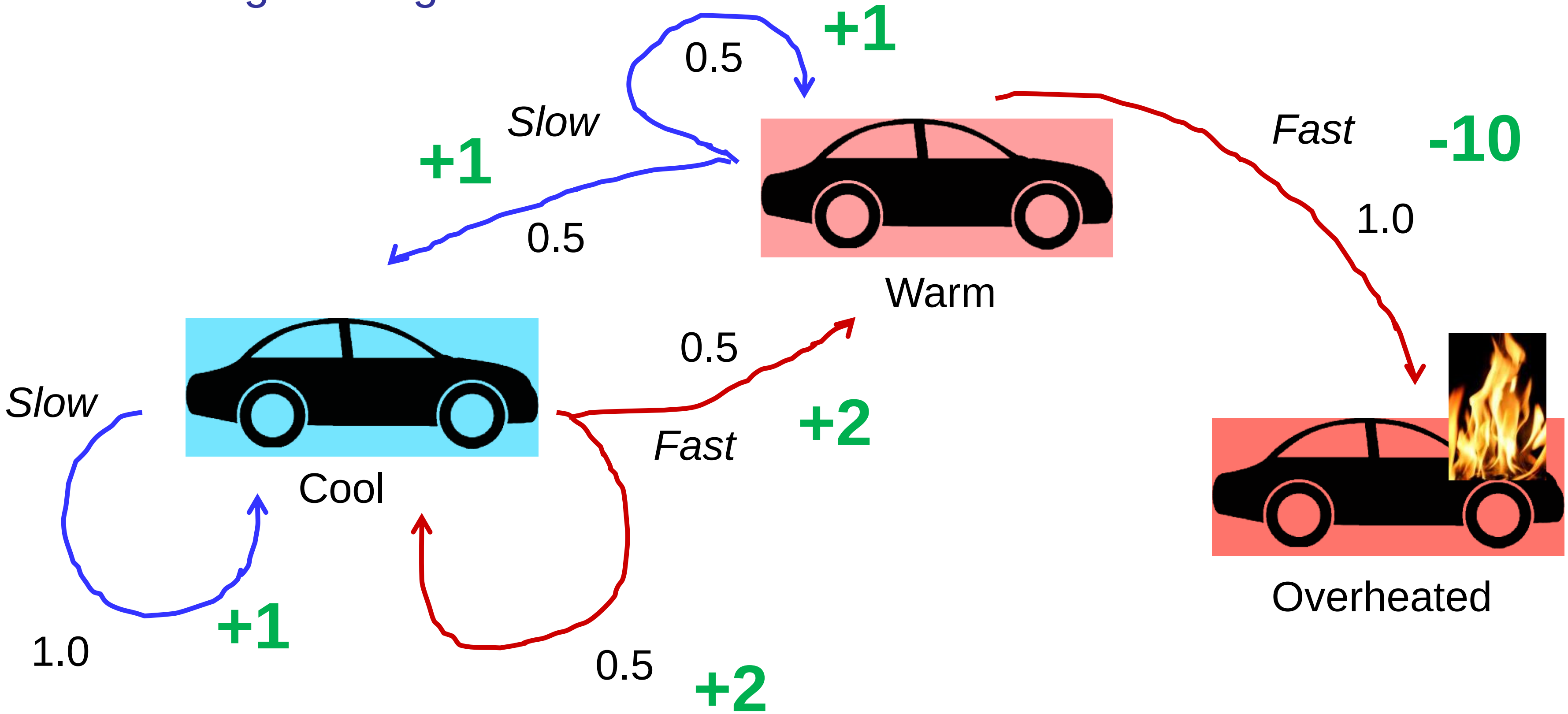


R(s) = -2.0

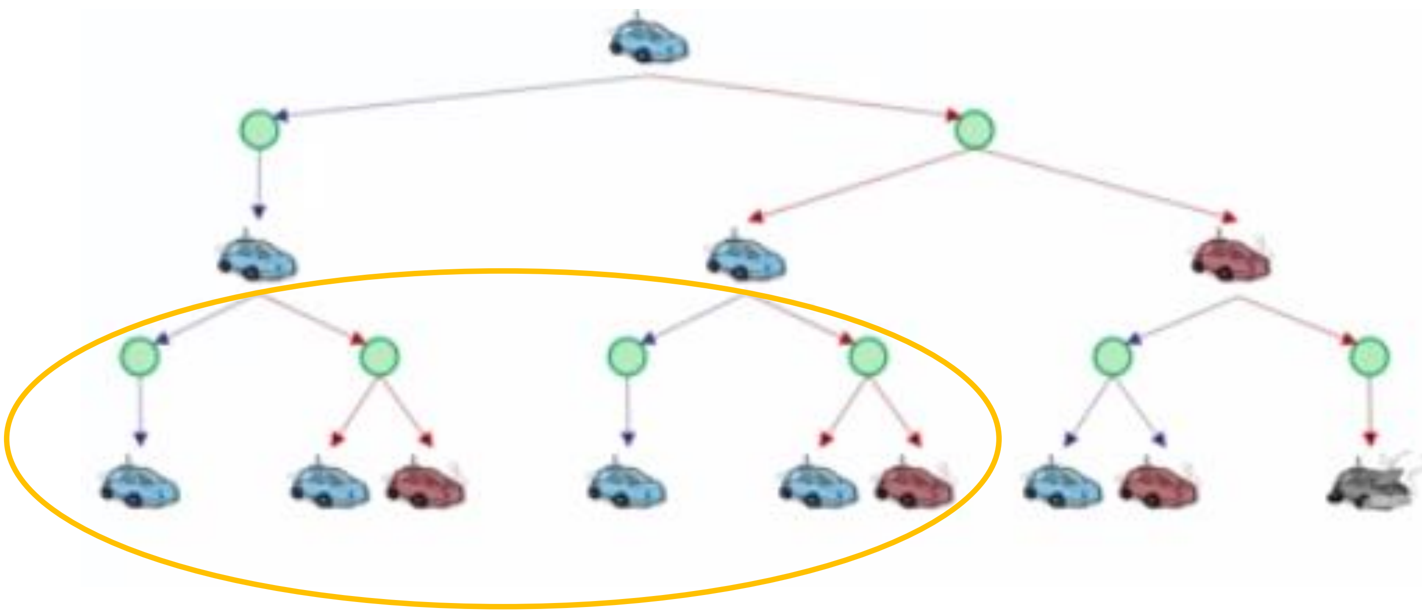
Example: Stuart Russell

Example: racing

- Robot car wants to travel far, quickly
- Three states: cool, warm, overheated
- Two actions: slow, fast
- Going faster gets double reward

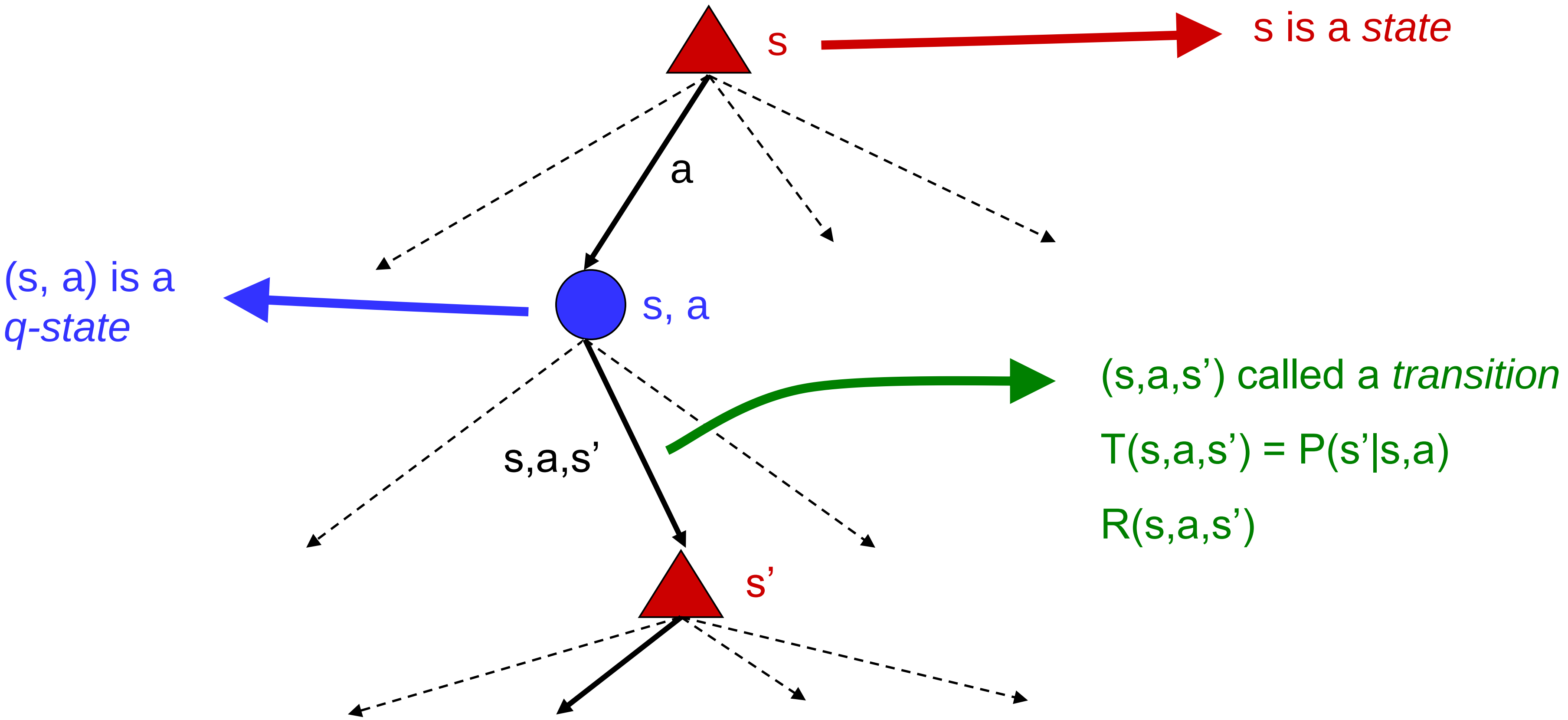


Racing search tree



MDP Search Trees

- Each MDP state projects an expectimax-like search tree



Utilities of sequences

- What preferences should an agent have over reward sequences?

- More or less? [1, 2, 2]   or   [2, 3, 4]

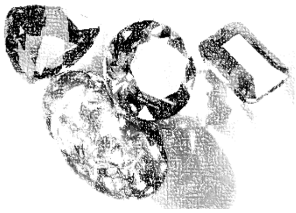
- Now or later?   [0, 0, 1]   or   [1, 0, 0]

Discounting

- It's reasonable to maximize the sum of rewards
- It's also reasonable to prefer rewards now to rewards later.
- One solution: value of rewards decay exponentially

1
Worth now

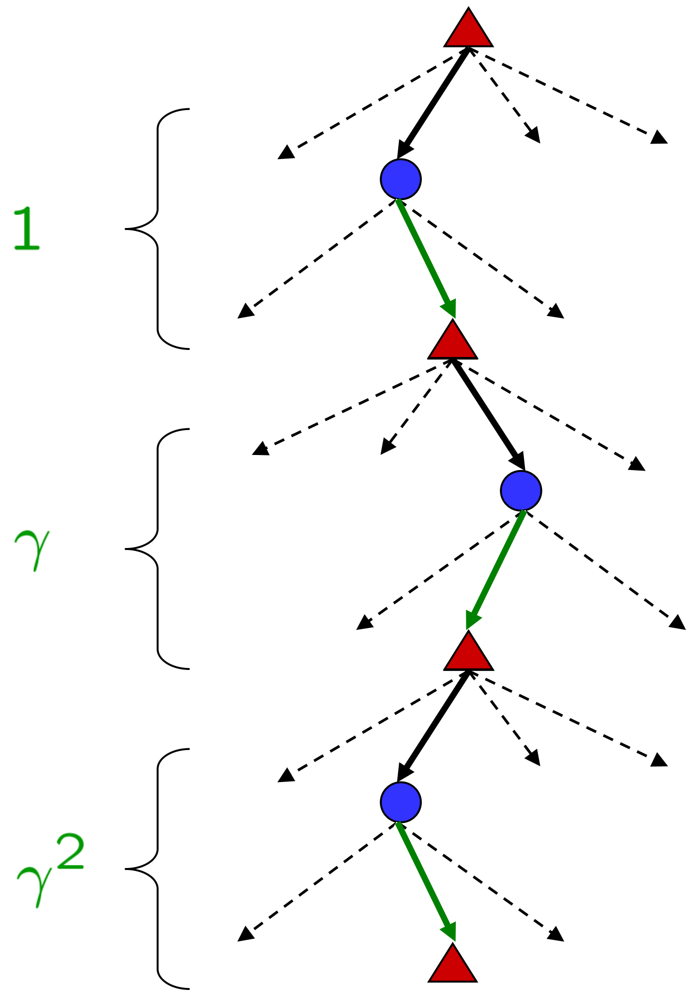
γ
Worth next step



γ^2
Worth in 2 steps

Discounting

- How to discount?
  - Each time we descend a level, we multiply in the discount once.
- Why discount?
  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge
- Example: discount of 0.5
  - U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3
  - U([1,2,3]) < U([3,2,1])



Stationary preferences

- What utility does a sequence of rewards have?
- Theorem: If we assume stationary preferences:

$$[r, r_0, r_1, r_2, \ldots] \succ [r, r'_0, r'_1, r'_2, \ldots]$$
$$\Leftrightarrow$$
$$[r_0, r_1, r_2, \ldots] \succ [r'_0, r'_1, r'_2, \ldots]$$

- Then: there are only two ways to define utilities
  - Additive utility:

$$U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$$

  - Discounted utility:

$$U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$$

Infinite Utilities?!

- Problem: infinite state sequences have infinite rewards

- Solutions:
  - Finite horizon (similar to depth-limited search):
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies (π depends on time left)

  - Discounting: for $0 < \gamma < 1$

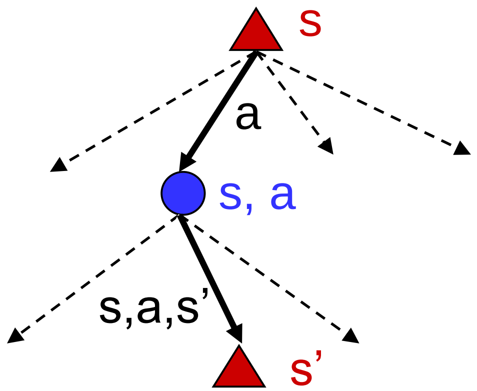
$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\text{max}}/(1-\gamma)$$

    - Smaller γ means smaller “horizon” – shorter term focus

  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like “overheated” for racing)

Recap: Defining MDPs

- Markov decision processes:
  - States S
  - Start state s_0
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)



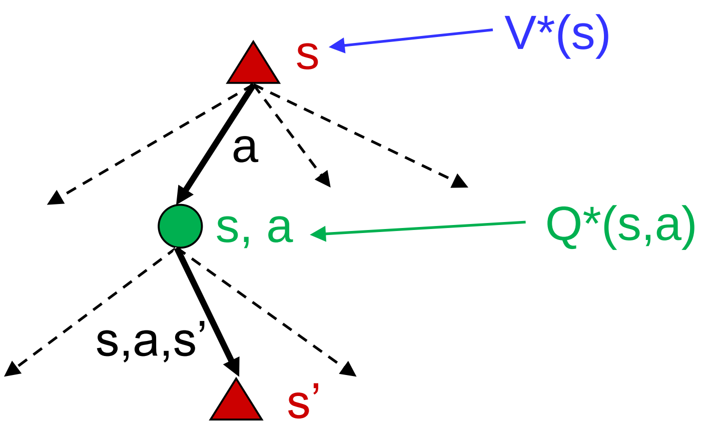
- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards

Optimal quantities

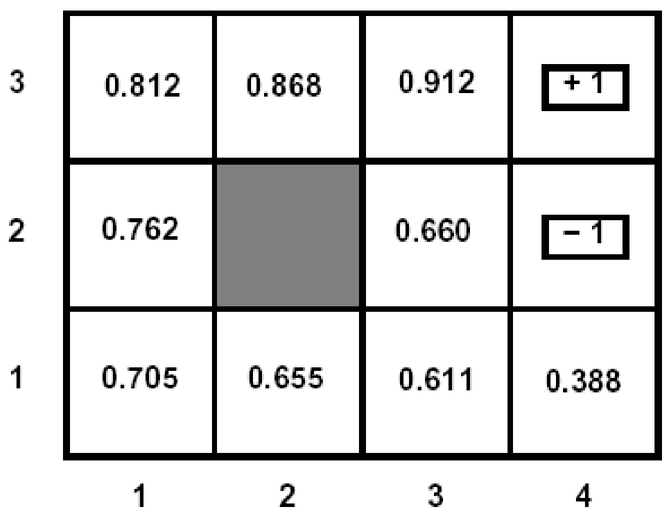
- Define the value (utility) of a state s:
  $V^*(s)$ = expected utility starting in s and acting optimally

- Define 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

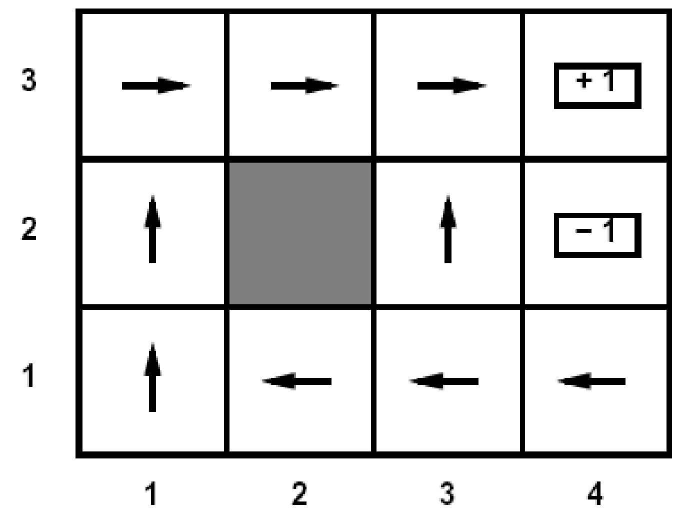
- Define the optimal policy:
  $\pi^*(s)$ = optimal action from state s



Gridworld example

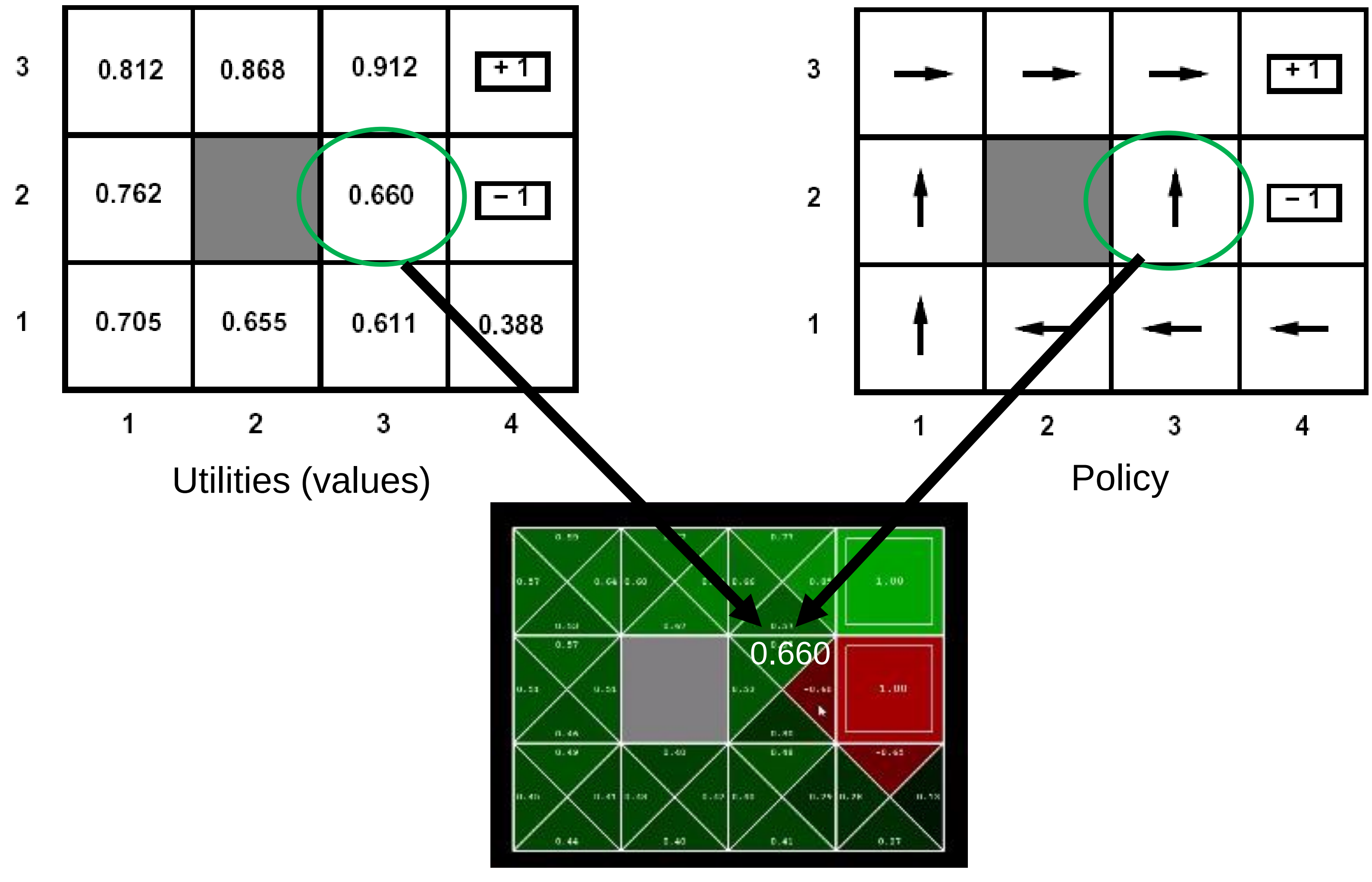


Utilities (values)



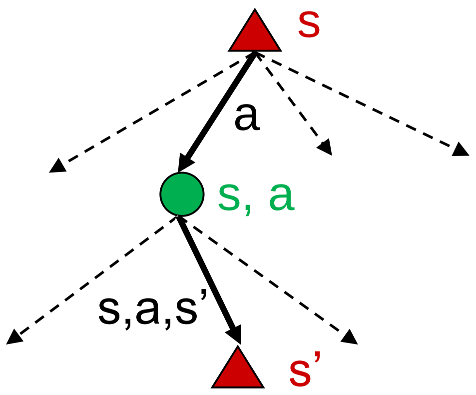
Policy

Gridworld example



Values of states: Bellman eqns

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax computed!



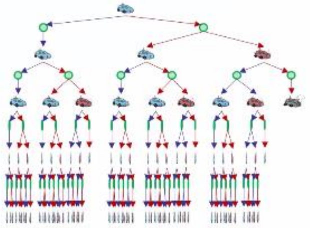
- Recursive definition of value:

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

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

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

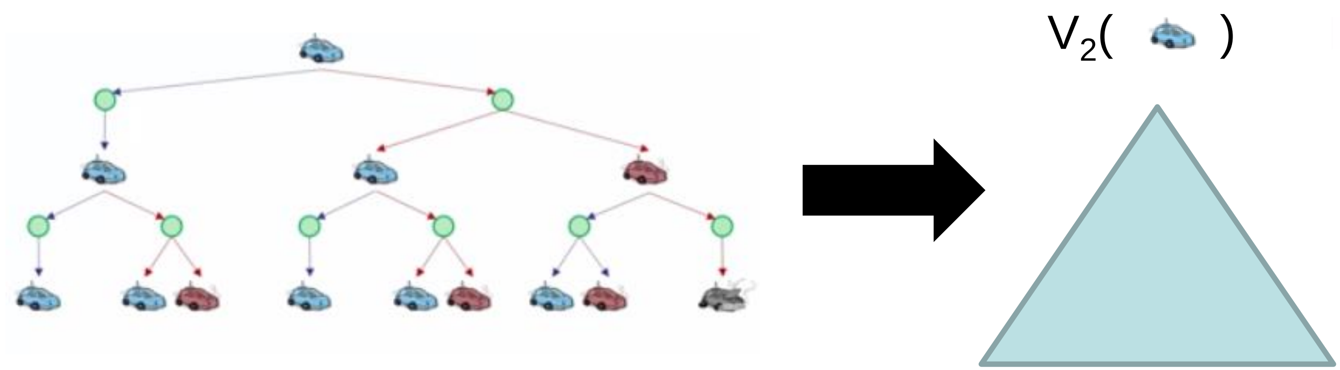
Recall: Racing search tree



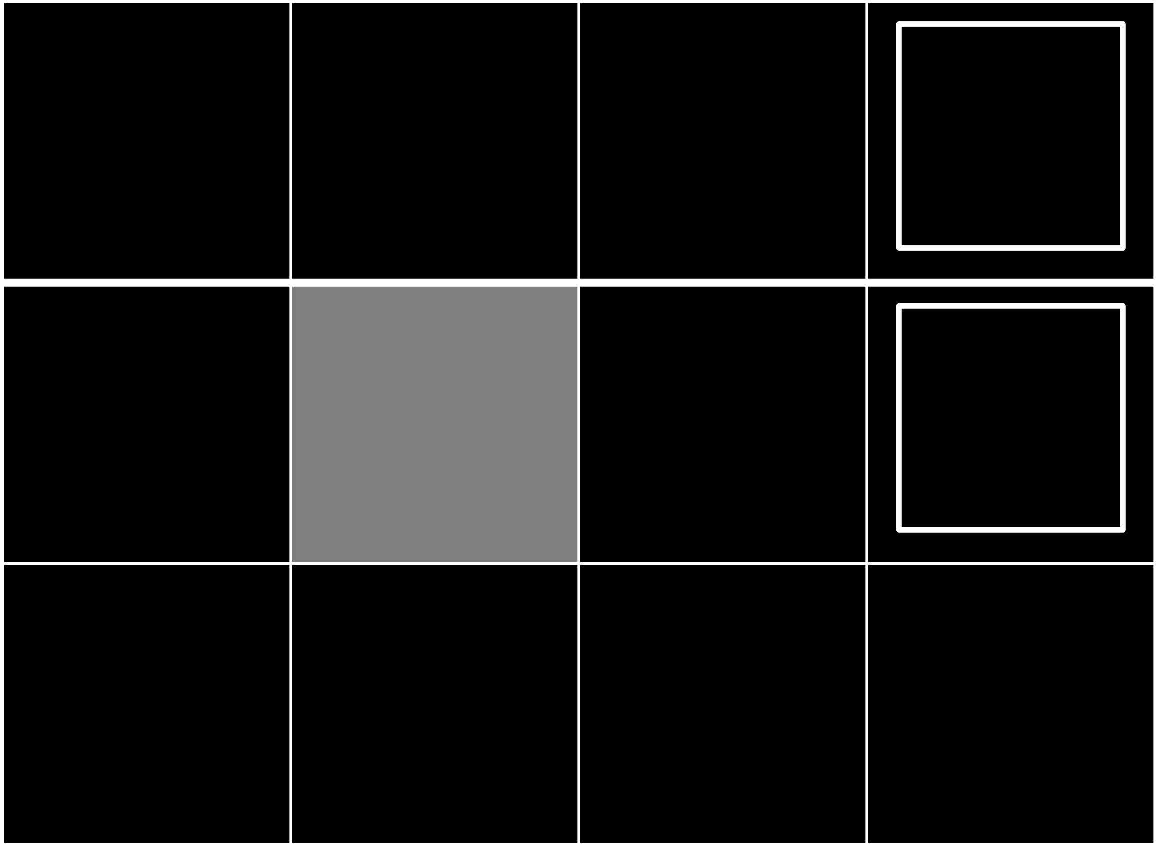
- We’re doing way too much work with expectimax!
- Problem: states are repeated
  - Idea: only compute needed quantities once
- Problem: tree goes on forever
  - Idea: do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree eventually don’t matter if $\gamma < 1$.

Time-limited values

- Key idea: time-limited values
- Define $V_k(s)$ to be the optimal value of *s* if the game ends in *k* more time steps.
  - Exactly what expectimax would give from *s*

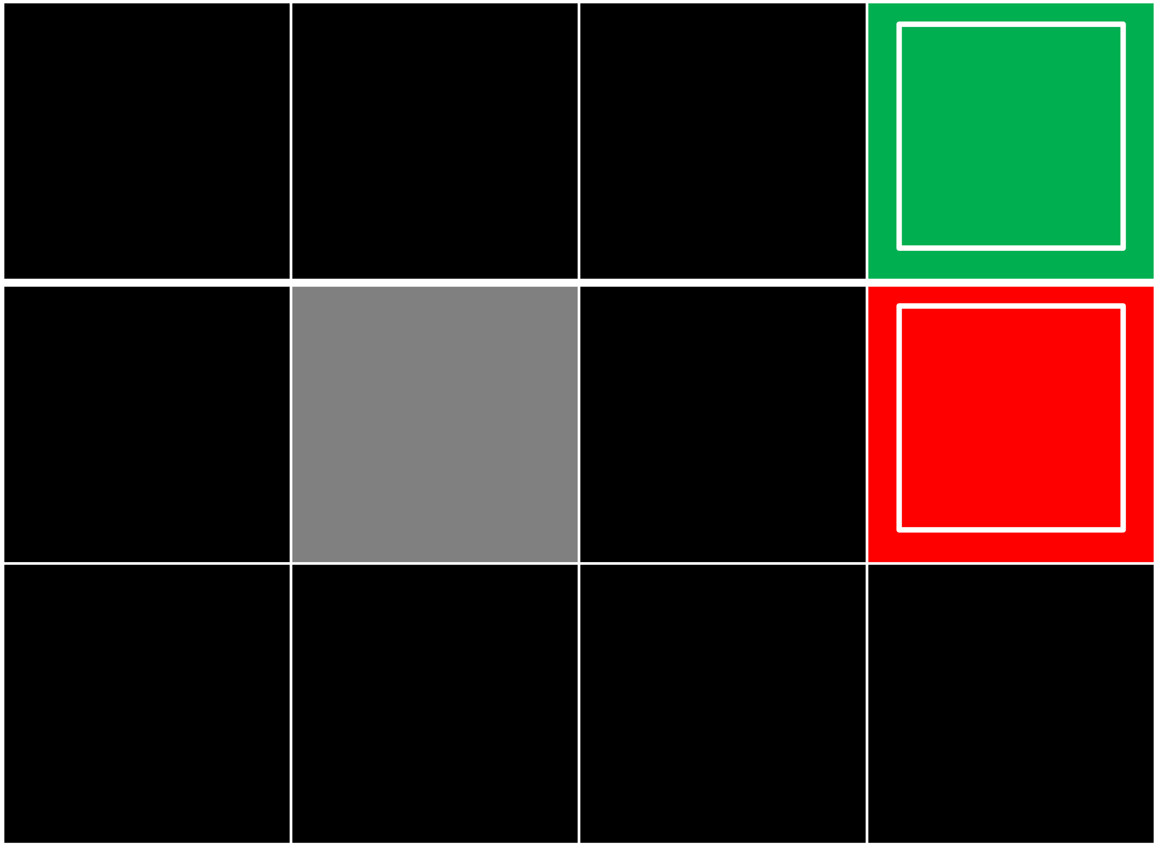


Gridworld example



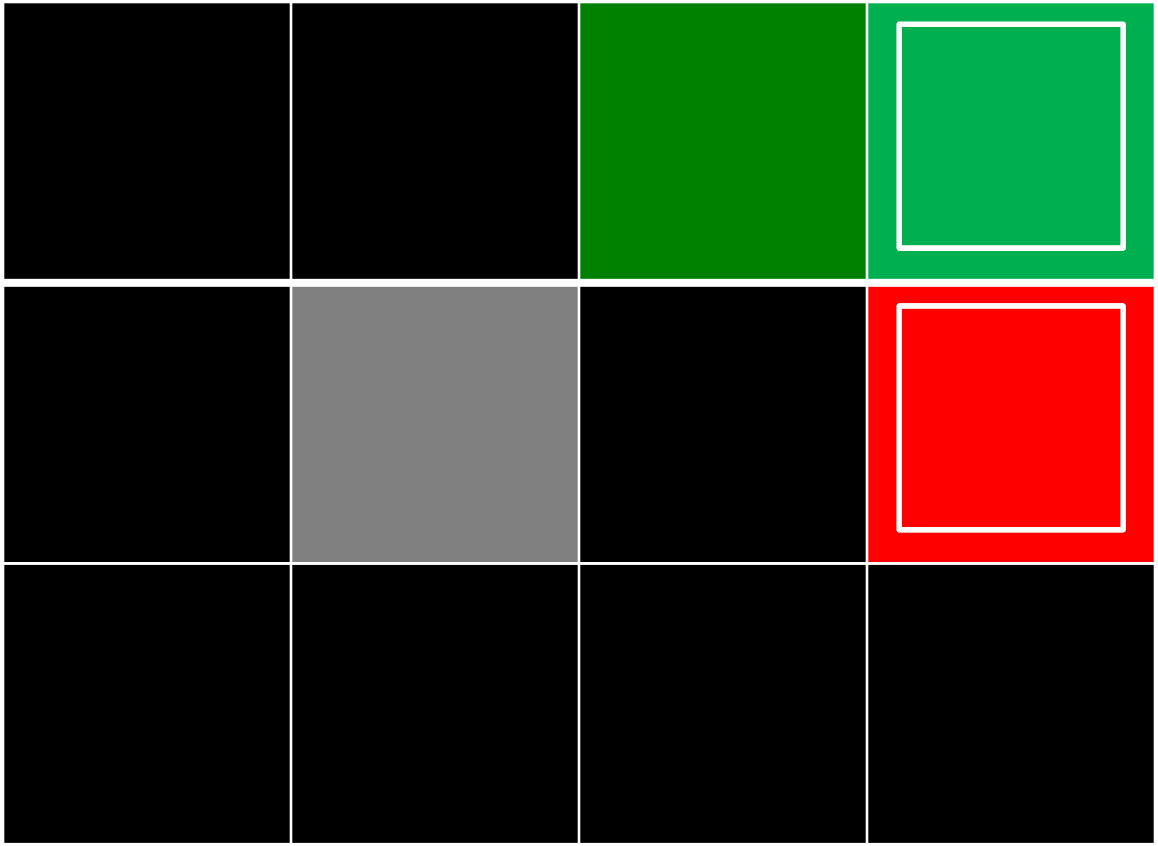
k=0 iterations

Gridworld example



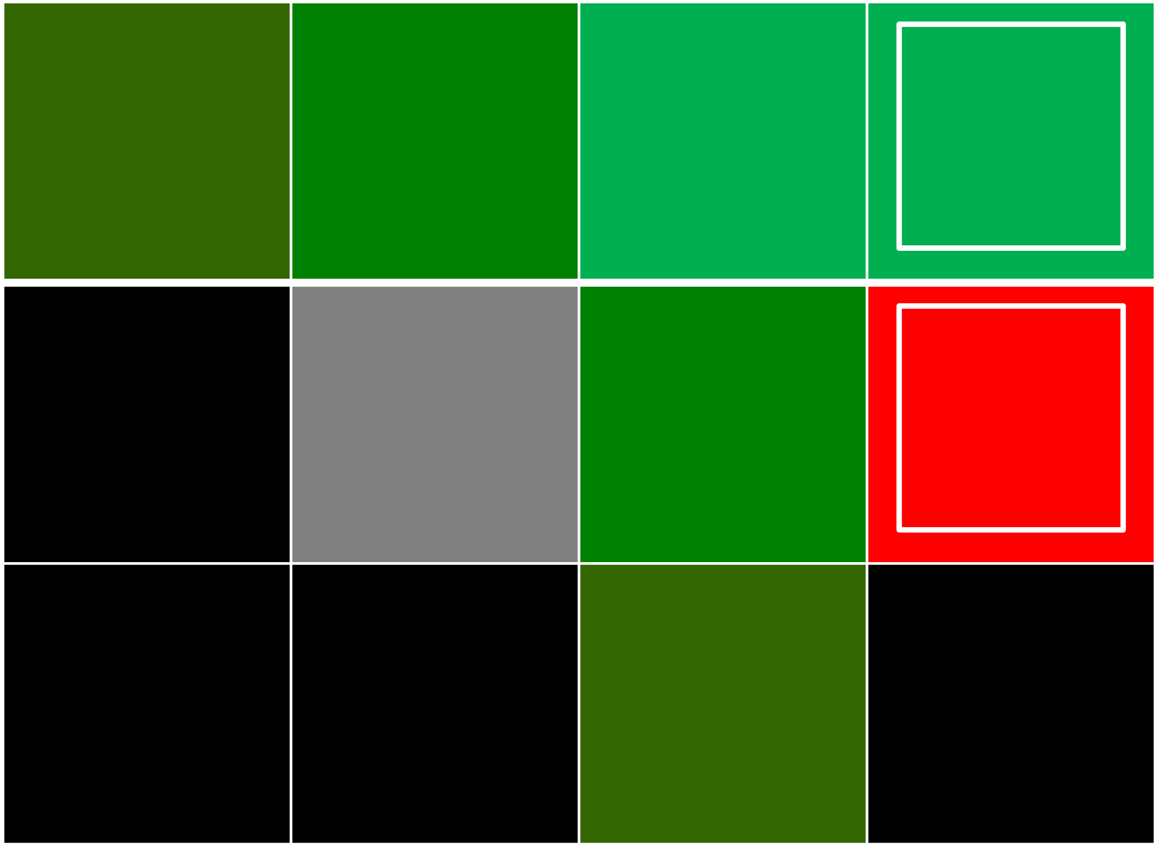
k=1 iterations

Gridworld example



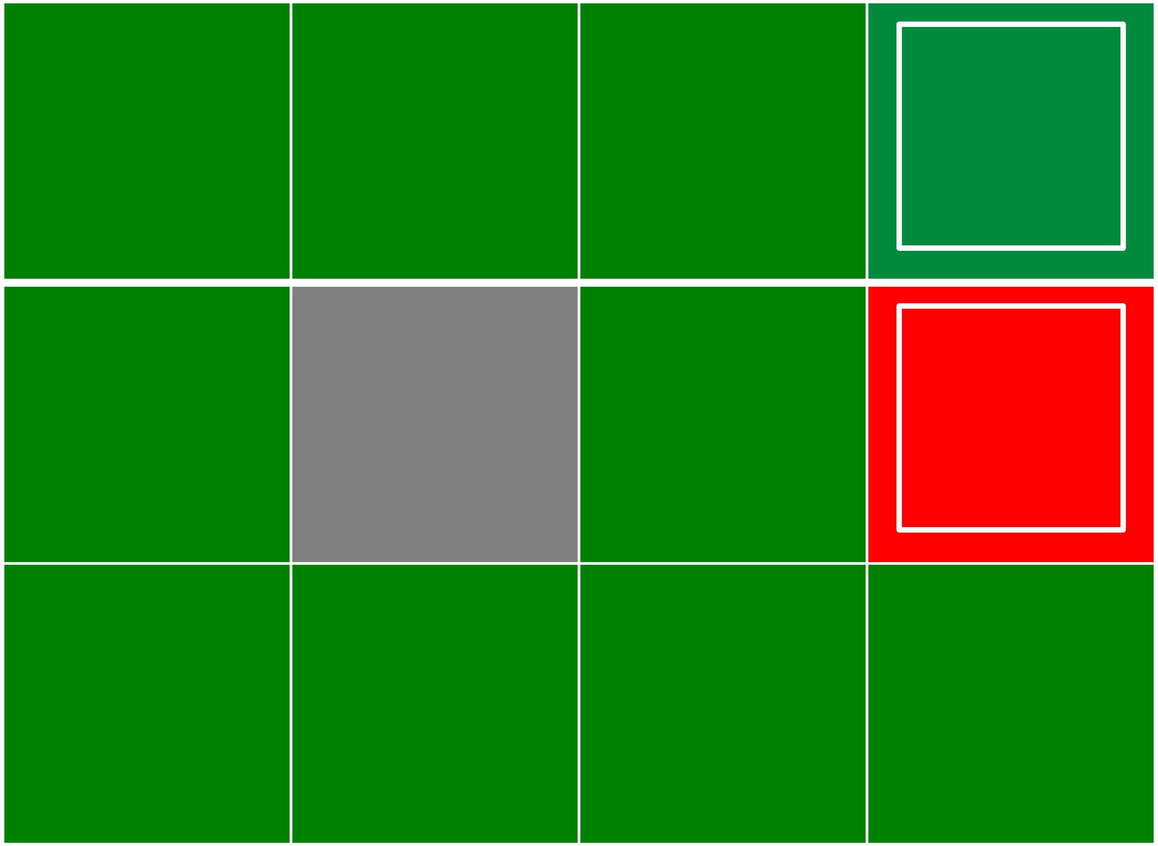
k=2 iterations

Gridworld example



k=3 iterations

Gridworld example



k=100 iterations

Computing time-limited values

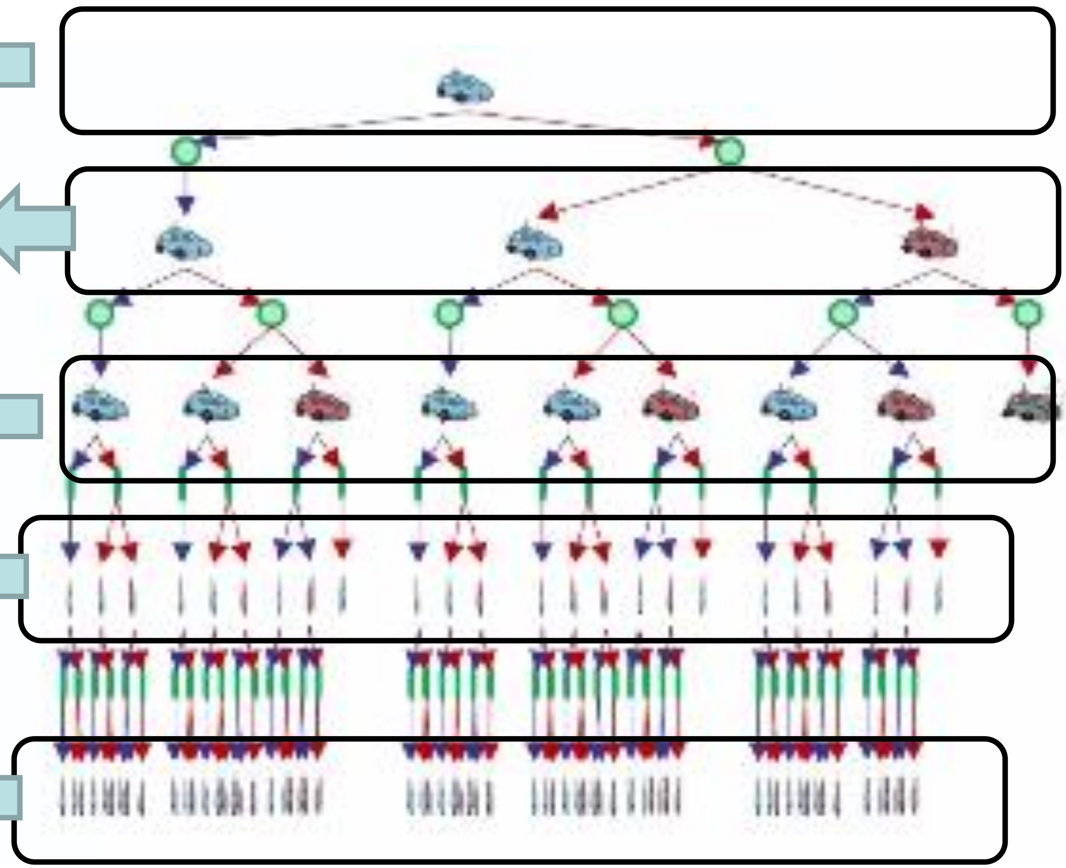
$V_4(\ )$ $V_4(\ )$ $V_4(\ )$

$V_3(\ )$ $V_3(\ )$ $V_3(\ )$

$V_2(\ )$ $V_2(\ )$ $V_2(\ )$

$V_1(\ )$ $V_1(\ )$ $V_1(\ )$

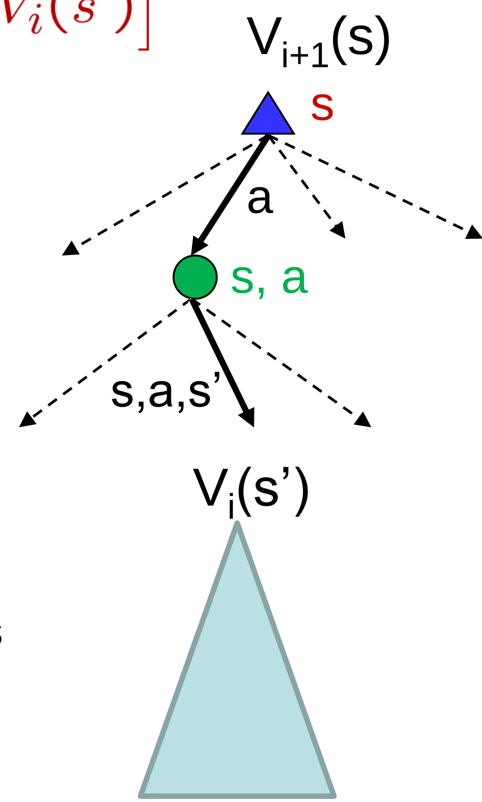
$V_0(\ )$ $V_0(\ )$ $V_0(\ )$



Value Iteration

- Start with $V_0^*(s) = 0$ for all s, which we know is right (why?).
- Given vector V_i^*, calculate the values for all states for depth i+1:

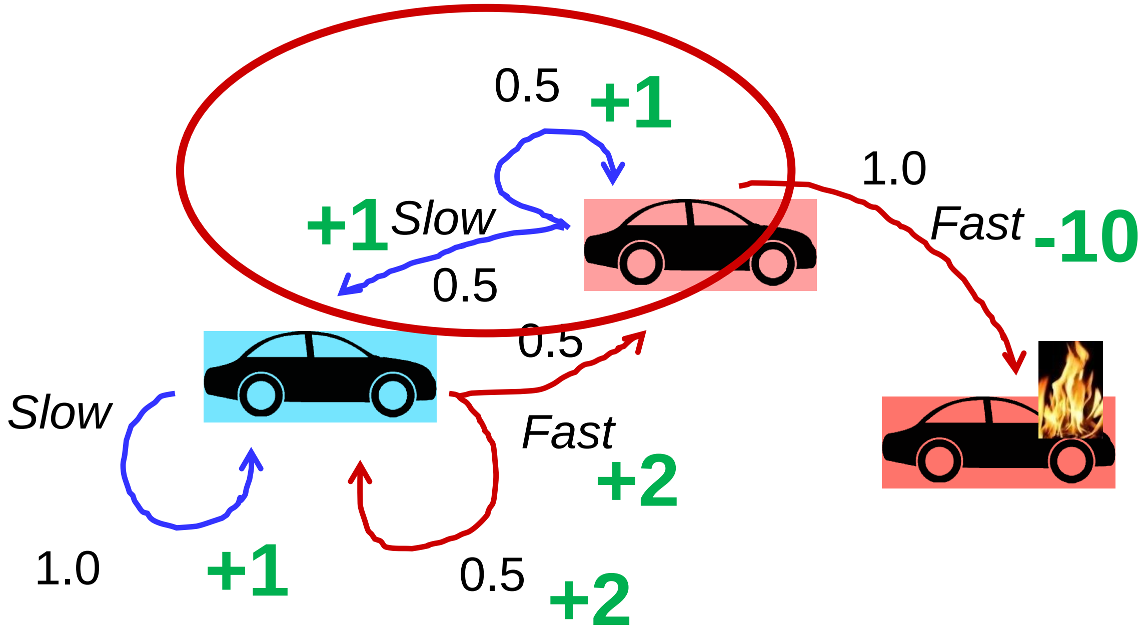
$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s')\right]$$



- Repeat until convergence
- This is called a value update or Bellman update
- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Note: Policy may converge long before values do.

Example: value iteration

V_2	Slow: 1+2 Fast: 2+0.5*2+0.5*1		0
V_1	2	1	0
V_0	0	0	0

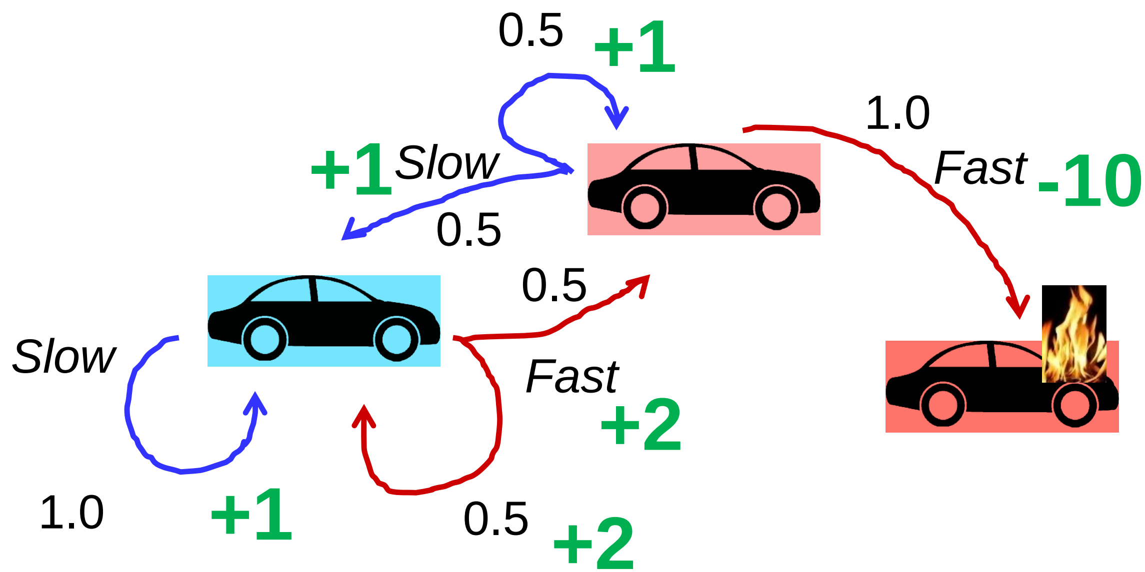


Assume no discount

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

Example: value iteration

V_2	3.5	?	0
V_1	2	1	0
V_0	0	0	0

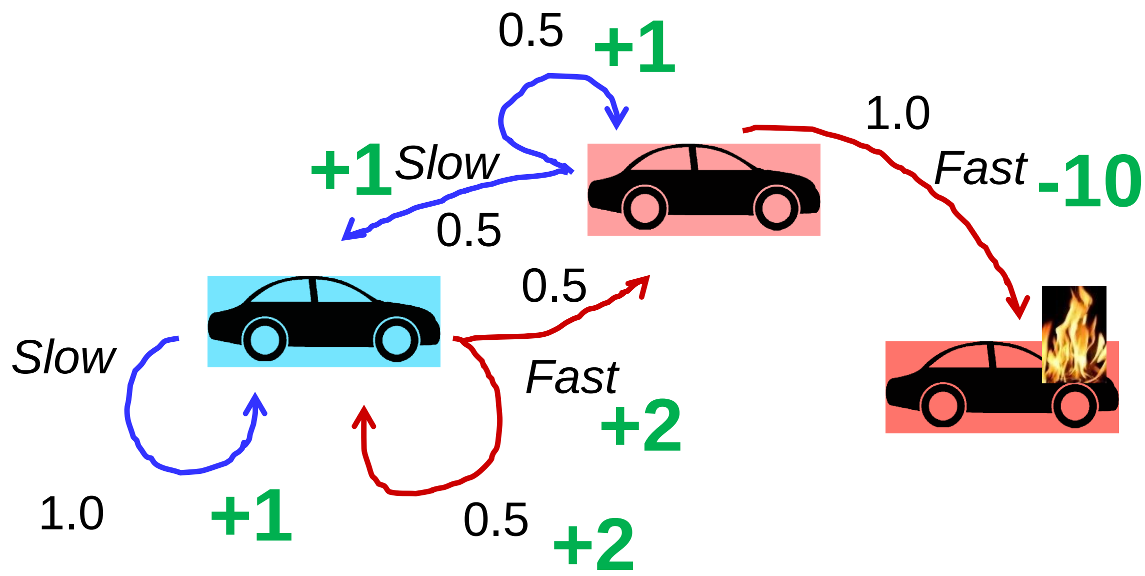


Assume no discount

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

Example: value iteration

V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0

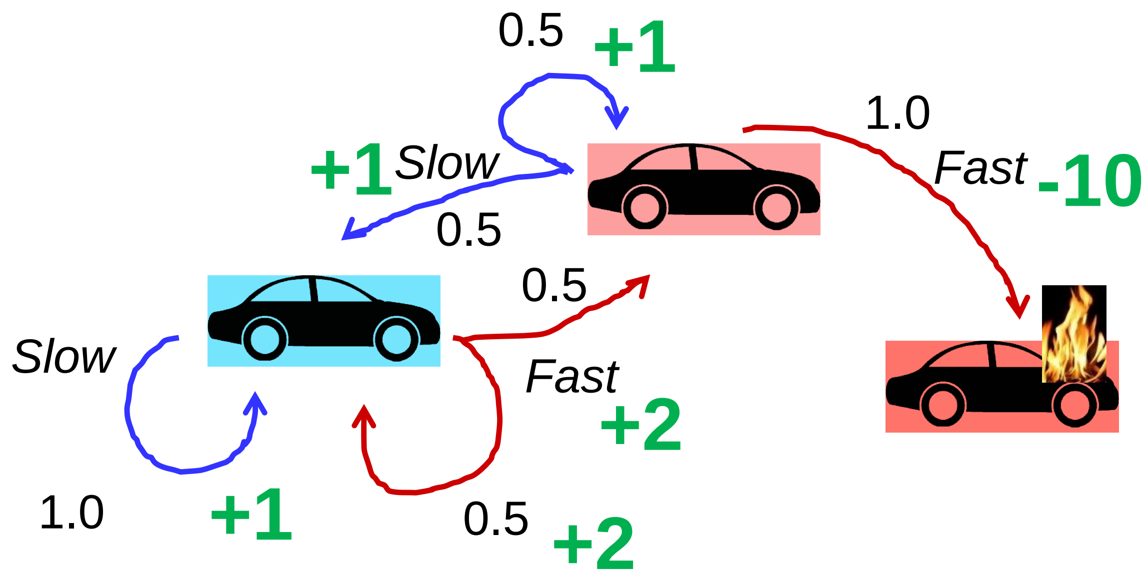


Assume no discount

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

Example: value iteration

V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0

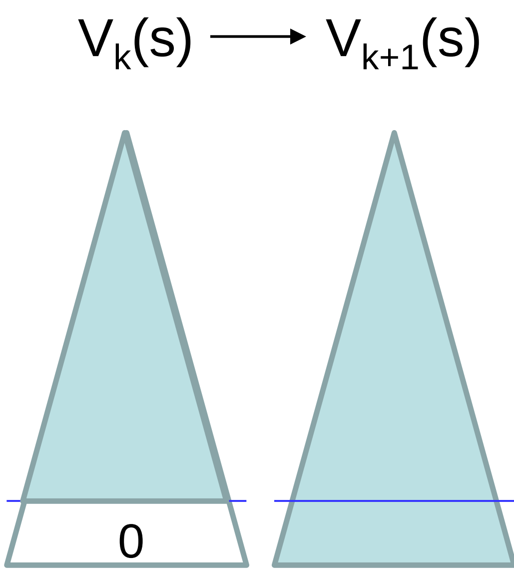


Assume no discount

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

Convergence

- Case 1: If the tree has maximum depth M, then V_M holds the actual untruncated values
- Case 2: If the discount is less than 1
  - Sketch: For any state, V_k and V_{k+1} can be viewed as depth k+1 expectimax resulting in nearly identical search trees.
  - The difference is that on the bottom layer, V_{k+1} has optimal rewards while V_k has zeros.
  - That last layer is at best all R_{MAX}
  - It is at worst R_{MIN}
  - But everything is discounted by γ^k that far out
  - So V_k and V_{k+1} are at most γ^k max|R| different
  - So as k increase, the values converge



Next time: policy-based methods