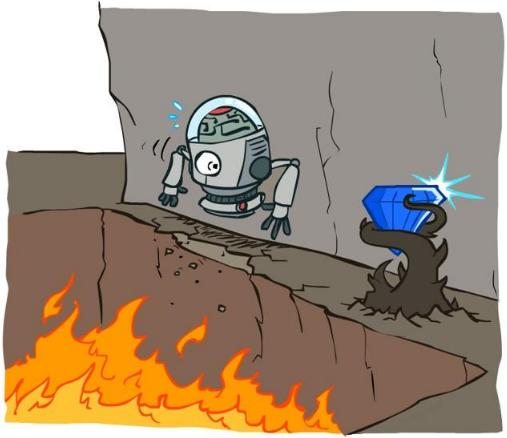
Artificial Intelligence

CSE 4617

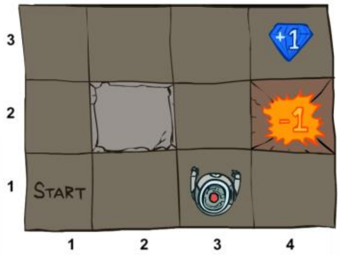
Ahnaf Munir
Assistant Professor
Islamic University of Technology

Non-Deterministic Search



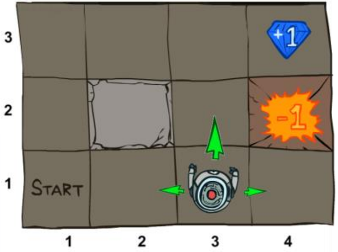
Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path



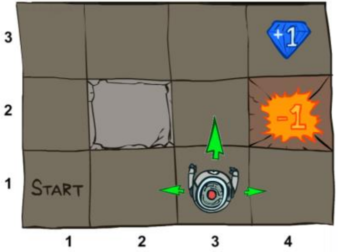
Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: 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



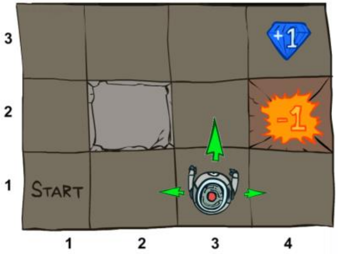
Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: 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
- The agent receives rewards each time step
  - Big rewards come at the end (good or bad)
  - Small "living" reward each step (can be negative)



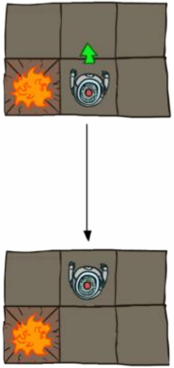
Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: 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
- The agent receives rewards each time step
  - Big rewards come at the end (good or bad)
  - Small "living" reward each step (can be negative)
- Goal: maximize sum of rewards



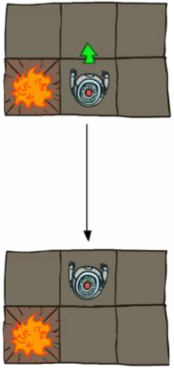
Grid World Actions

Deterministic Grid World

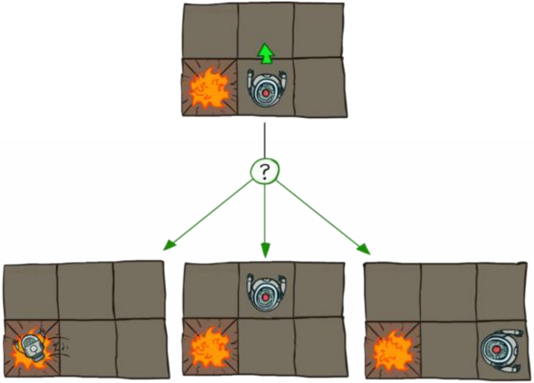


Grid World Actions

Deterministic Grid World

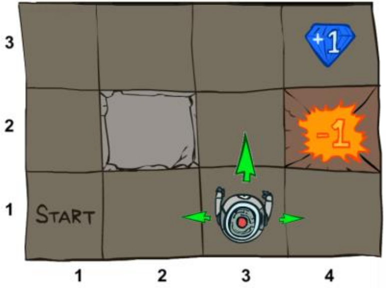


Stochastic Grid World



Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - 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
  - A reward function $R(s, a, s')$
    - Sometimes just $R(s)$ or $R(S')$
  - A start state
  - Maybe a terminal state
- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search
  - 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

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

- This is just like search, where the successor function could only depend on the current state (not the history)

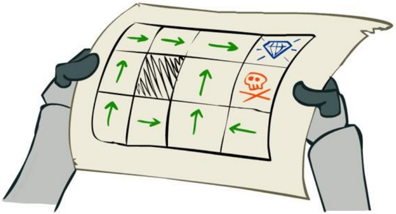
Andrey Markov
(1856-1922)

Policies

- In deterministic single-agent search problems, we wanted an optimal **plan**, or sequence of actions, from start to a goal

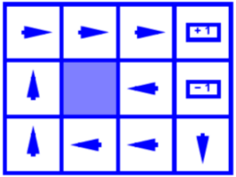
-

  For MDPs, we want an optimal **policy** $\pi^* : S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed



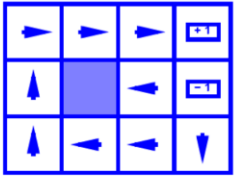
Optimal policy when $R(s, a, s') = -0.04$ for all non-terminal s

Optimal Policies

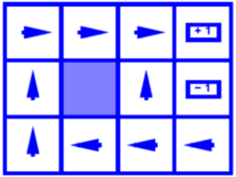


R(s) = -0.01

Optimal Policies

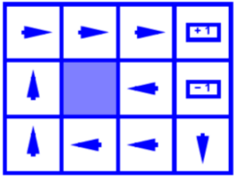


R(s) = -0.01

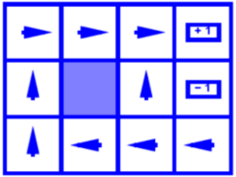


R(s) = -0.03

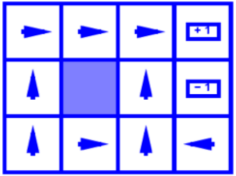
Optimal Policies



R(s) = -0.01

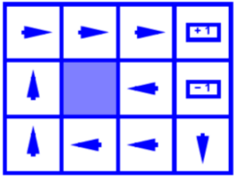


R(s) = -0.03

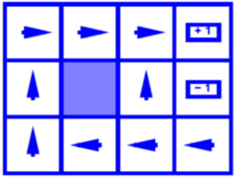


R(s) = -0.4

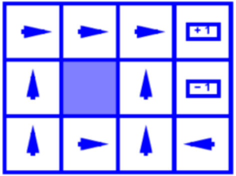
Optimal Policies



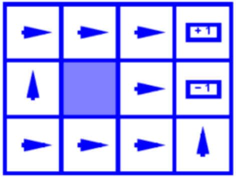
R(s) = -0.01



R(s) = -0.03

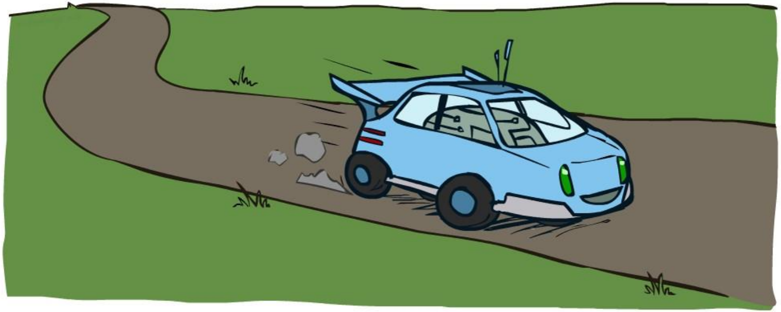


R(s) = -0.4



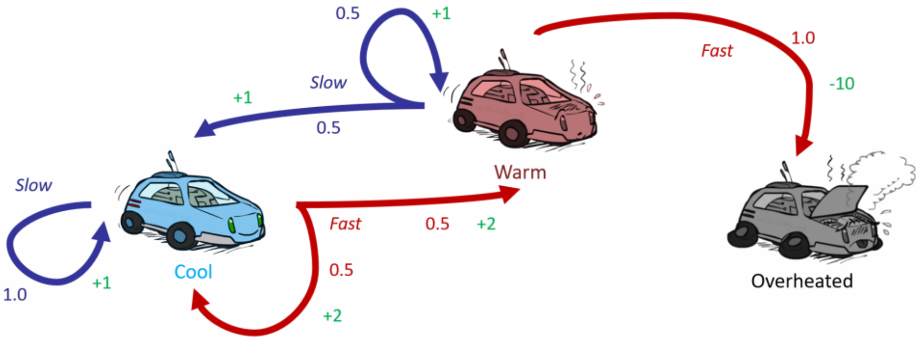
R(s) = -2.0

Example: Racing



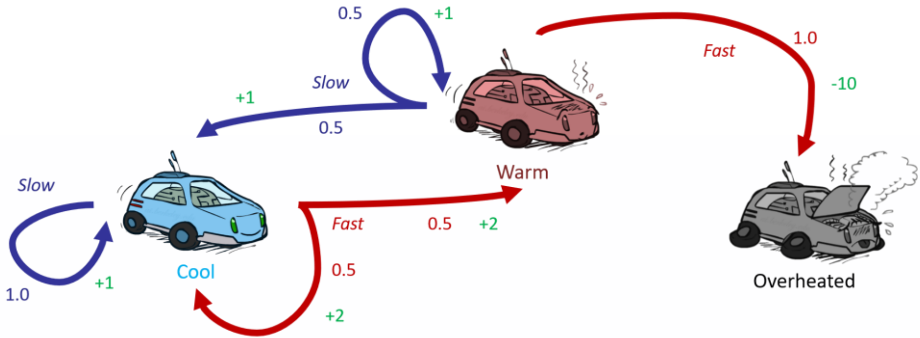
Example: Racing

- A robot car wants to travel far, quickly



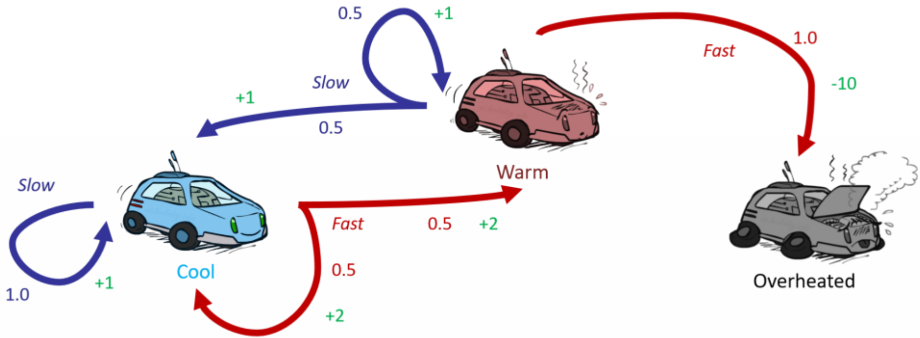
Example: Racing

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated



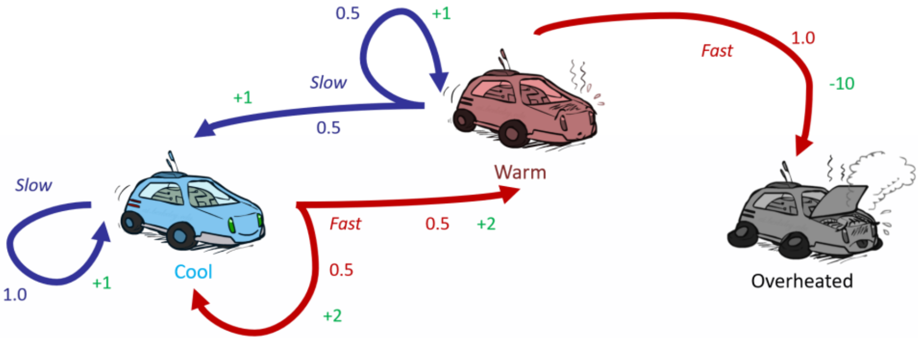
Example: Racing

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*



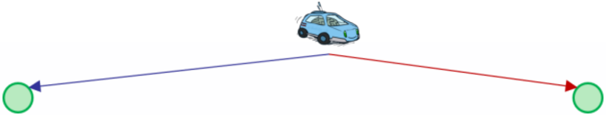
Example: Racing

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward

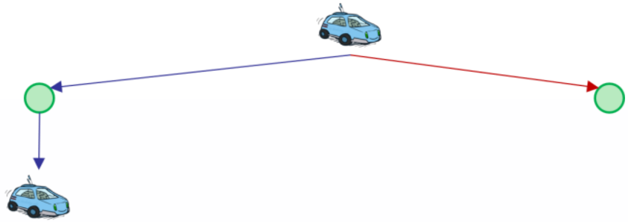


Racing Search Tree

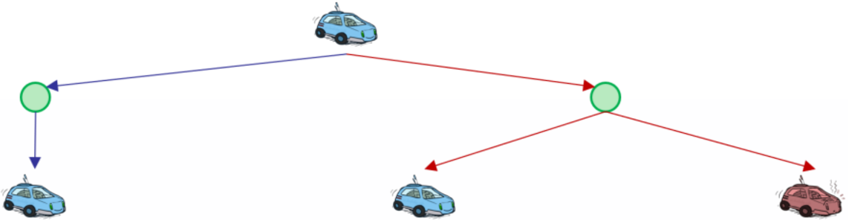
Racing Search Tree



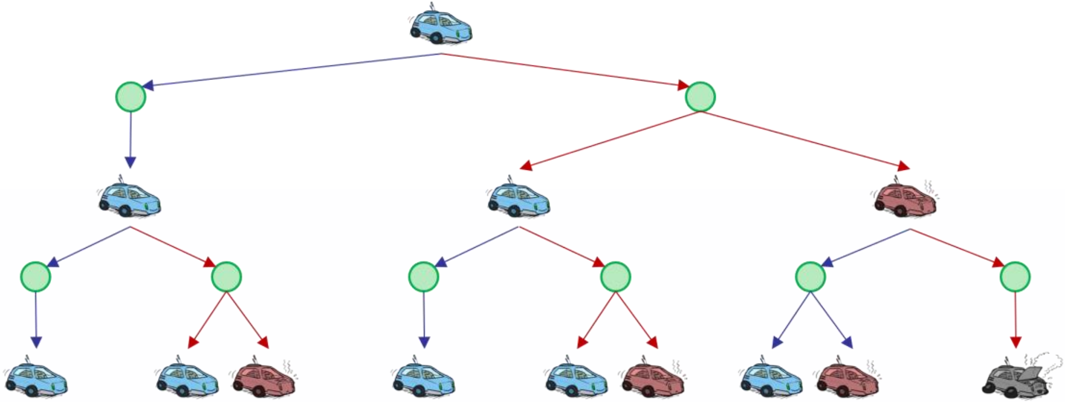
Racing Search Tree



Racing Search Tree

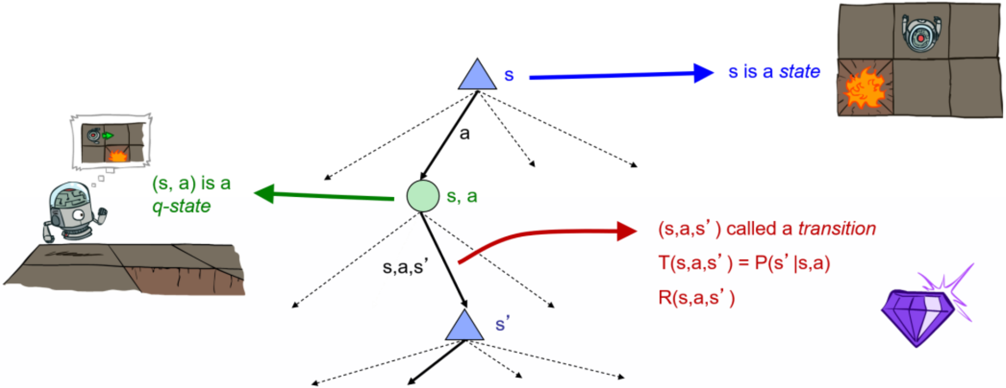


Racing Search Tree

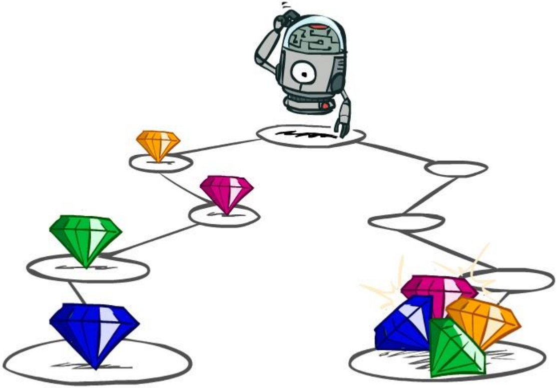


MDP Search Trees

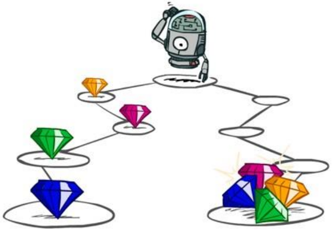
Each MDP state projects an expectimax-like search tree



Utilities of Sequences

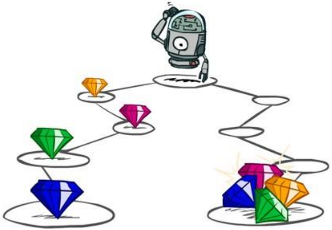


Utilities of Sequences



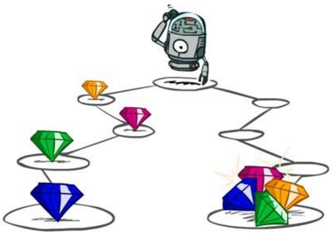
- What preferences should an agent have over reward sequences?
- More or less?

Utilities of Sequences



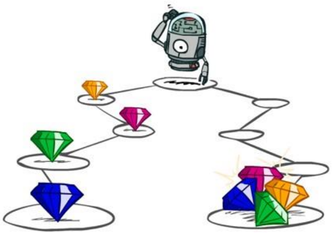
- What preferences should an agent have over reward sequences?
- More or less? $[1, 2, 2]$ or $[2, 3, 4]$

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?

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: values of rewards decay exponentially

Discounting

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

1

Worth Now

Discounting

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

1

Worth Now

γ

Worth Next Step

Discounting

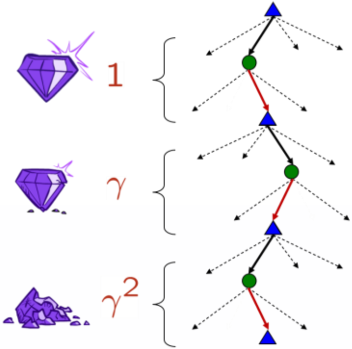
- It's reasonable to maximize the sum of rewards
- It's also reasonable to prefer rewards now to rewards later
- One solution: values of rewards decay exponentially

1	γ	γ^2
Worth Now	Worth Next Step	Worth In Two Steps

Discounting

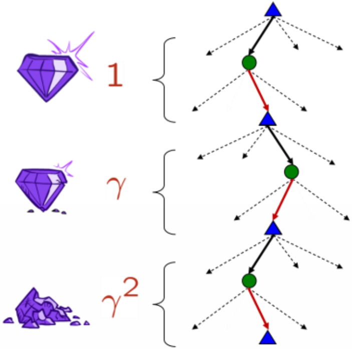
- How to discount?

- Why discount?



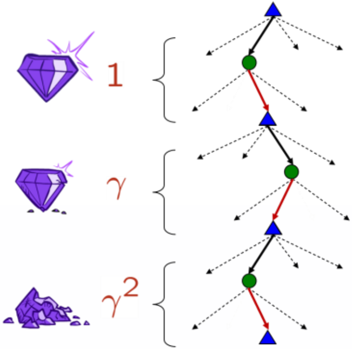
Discounting

- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?



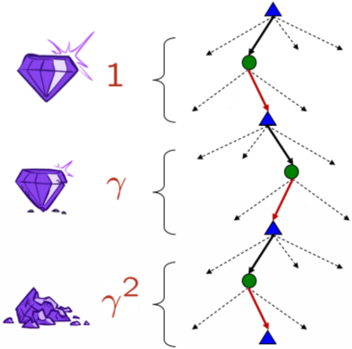
Discounting

- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards



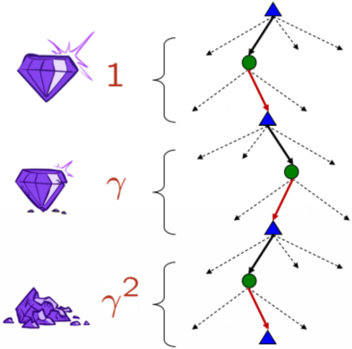
Discounting

- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards
- Example: discount of 0.5



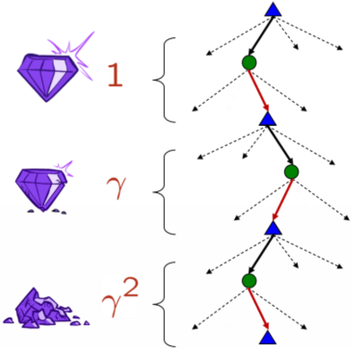
Discounting

- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards
- Example: discount of 0.5
  - $U([1,2,3]) = 1\times1+0.5\times2+0.25\times3$



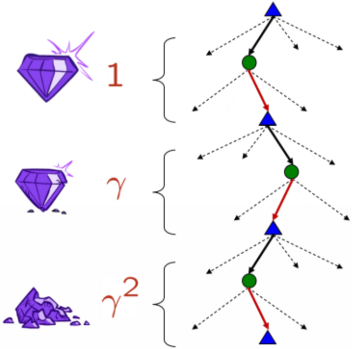
Discounting

- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards
- Example: discount of 0.5
  - $U([1,2,3]) = 1\times1+0.5\times2+0.25\times3$
  - $U([3,2,1]) = 3\times1+0.5\times2+0.25\times1$

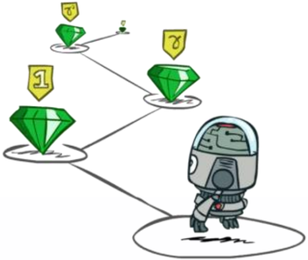


Discounting

- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards
- Example: discount of 0.5
  - $U([1,2,3]) = 1\times1+0.5\times2+0.25\times3$
  - $U([3,2,1]) = 3\times1+0.5\times2+0.25\times1$
  - $U([1,2,3]) < U([3,2,1])$



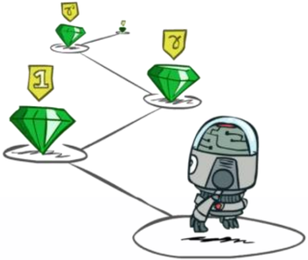
Stationary Preferences



Stationary Preferences

- Theorem: If we assume **stationary preferences:**

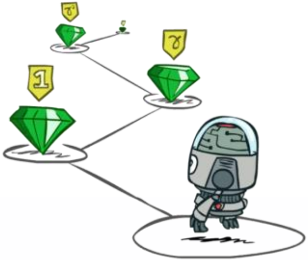
$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$
$$\Updownarrow$$
$$[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$



Stationary Preferences

- Theorem: If we assume **stationary preferences**:

$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$
$$\Updownarrow$$
$$[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$



- Then: there are only two ways to define utilities
  - Additive utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \ldots$
  - Discounted utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$

Quiz: Discounting

- Given:
  - Actions: East, West, and Exit (only available in exit states a, e)
  - Transitions: deterministic

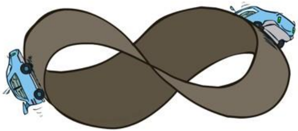
10				1
a	b	c	d	e

10				1

- Quiz 1: For $\gamma = 1$, what is the optimal policy?
- Quiz 2: For $\gamma = 0.1$, what is the optimal policy?
- Quiz 3: For which γ are West and East equally good when in state d?

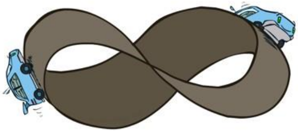
Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?



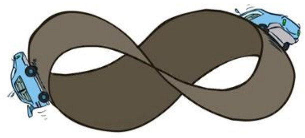
Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solution:
  - Finite horizon: (similar to depth-limited search)
  - Discounting: $0 < \gamma < 1$
  - Absorbing state



Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solution:
  - 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: $0 < \gamma < 1$



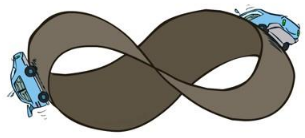
  - Absorbing state

Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solution:
  - 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: $0 < \gamma < 1$

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

    - Smaller γ means smaller "horizon" - shorter term focus
  - Absorbing state

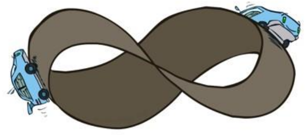


Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solution:
  - 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: $0 < \gamma < 1$

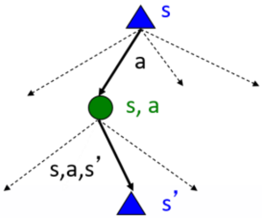
$$U([r_0, \ldots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1-\gamma)$$

    - 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)

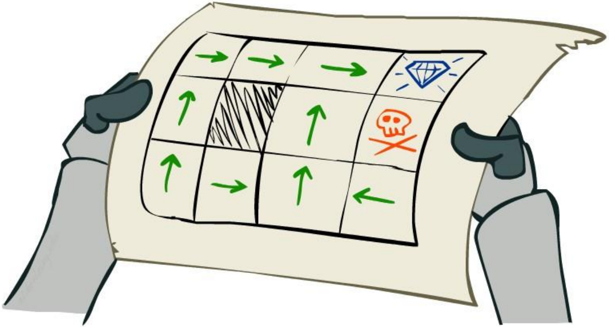


Recap: Defining MDPs

- Markov Decision Processes
  - Set of states S
  - Start state s_0
  - Set of 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

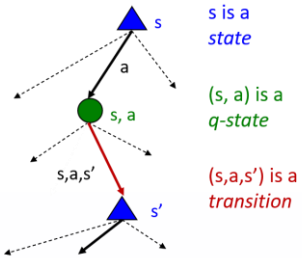


Solving MDPs

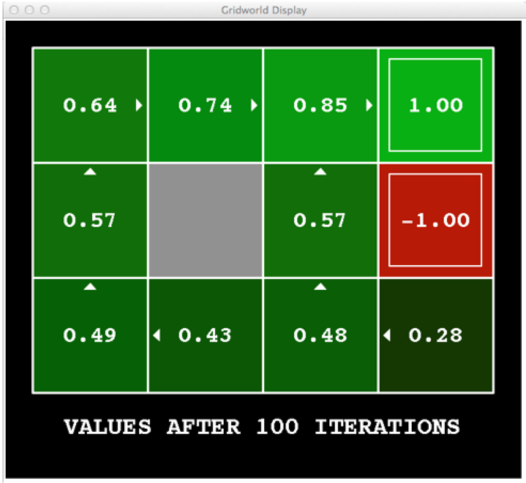


Optimal Quantities

- 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

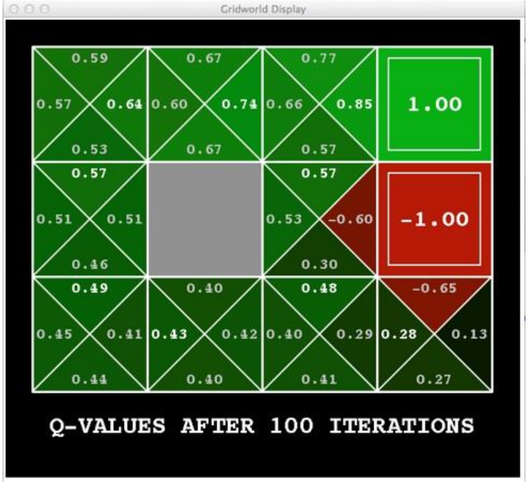


Snapshot of Demo Gridworld Values



Noise: 0.2
Discount: 0.9
Living reward: 0

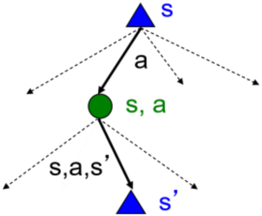
Snapshot of Demo Gridworld Values



Noise: 0.2
Discount: 0.9
Living reward: 0

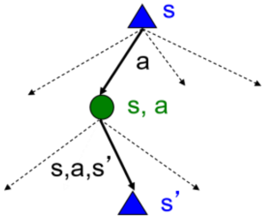
Values of States

- 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!



Values of States

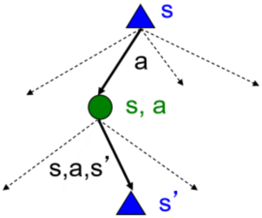
- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax computed!



- Recursive definition of value:

Values of States

- 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!

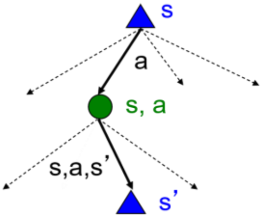


- Recursive definition of value:

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

Values of States

- 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!



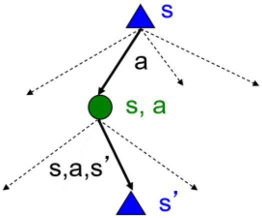
- Recursive definition of value:

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

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

Values of States

- 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:

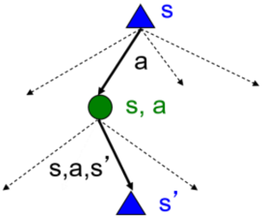
$$V^*(s) = \max_a Q^*(s,a)$$

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

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

Values of States

- 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!



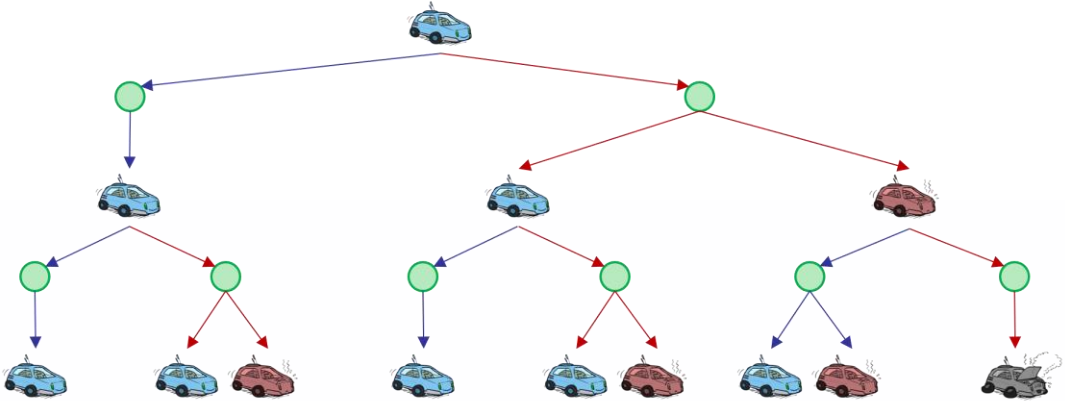
- Recursive definition of value:

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

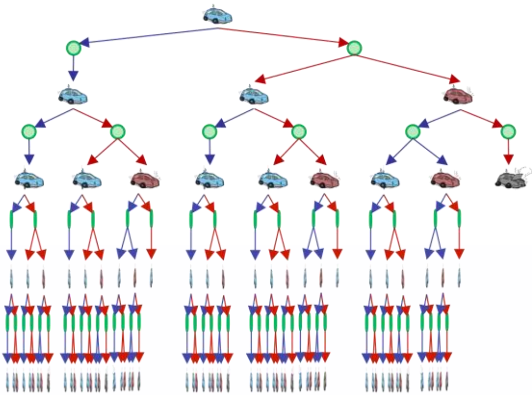
$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

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

Racing Search Tree

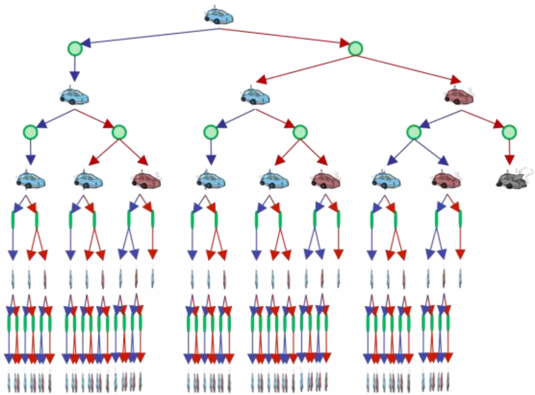


Racing Search Tree



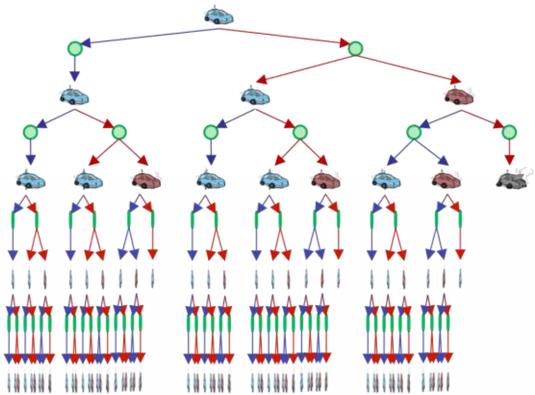
Racing Search Tree

- We're doing way too much work with expectimax!



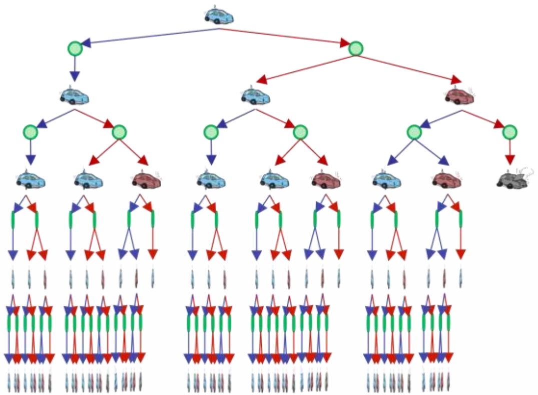
Racing Search Tree

- We're doing way too much work with expectimax!
- Problem: States are repeated



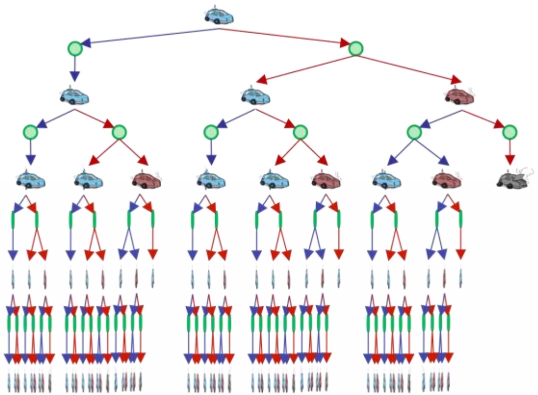
Racing Search Tree

- We're doing way too much work with expectimax!
- Problem: States are repeated
- Problem: Tree goes on forever



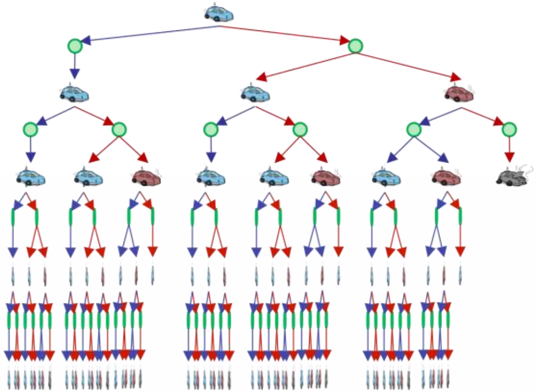
Racing Search Tree

- We're doing way too much work with expectimax!
- Problem: States are repeated
  - Idea: Only compute needed quantities once
- Problem: Tree goes on forever



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

Time-Limited Values

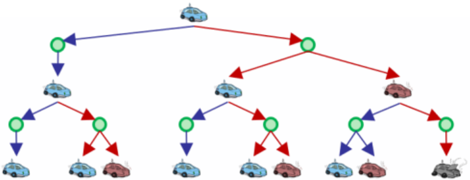
- Key idea: time-limited values
- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps

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
  - Equivalently, it's what a depth-k expectimax would give from s

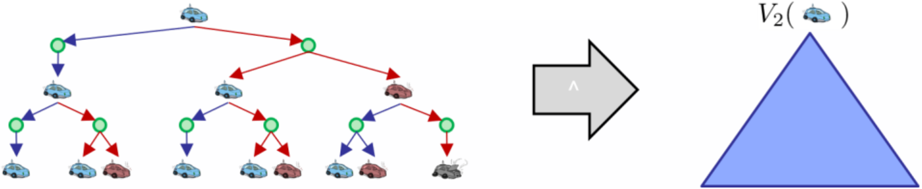
Time-Limited Values

- 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
  - Equivalently, it's what a depth-k expectimax would give from s



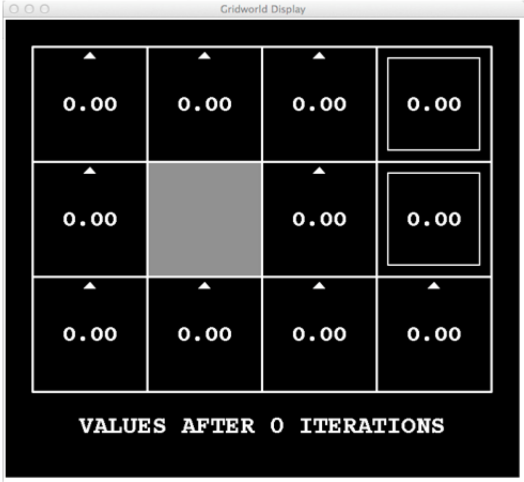
Time-Limited Values

- 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
  - Equivalently, it's what a depth-k expectimax would give from s



Example

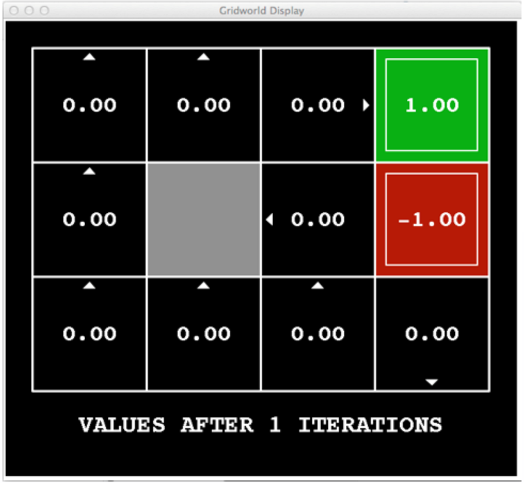
k = 0



Noise = 0.2
Discount = 0.9
Living reward = 0

Example

k = 1



Noise = 0.2
Discount = 0.9
Living reward = 0

Example

k = 2

Noise = 0.2
Discount = 0.9
Living reward = 0

Example

k = 3

Noise = 0.2
Discount = 0.9
Living reward = 0

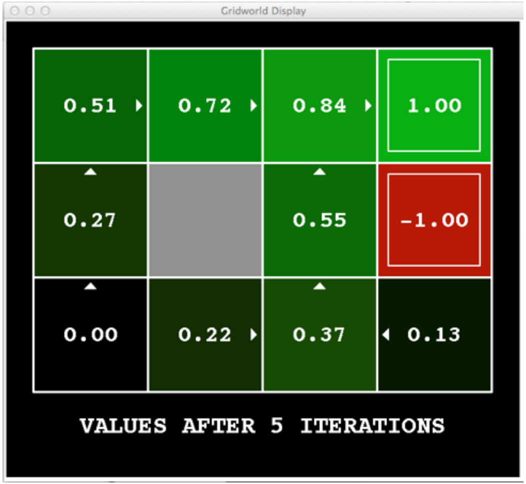
Example

k = 4

Noise = 0.2
Discount = 0.9
Living reward = 0

Example

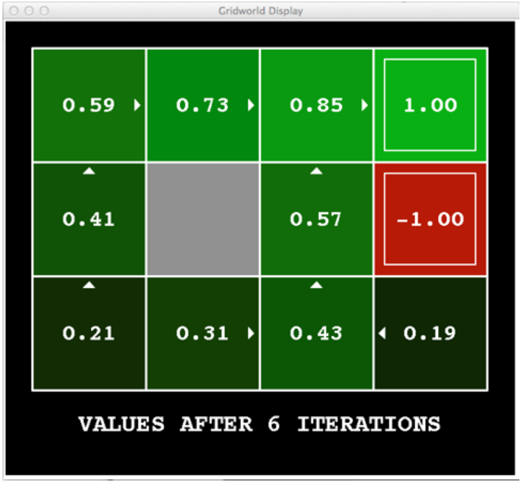
k = 5



Noise = 0.2
Discount = 0.9
Living reward = 0

Example

k = 6



Noise = 0.2
Discount = 0.9
Living reward = 0

Example

k = 7

Noise = 0.2
Discount = 0.9
Living reward = 0

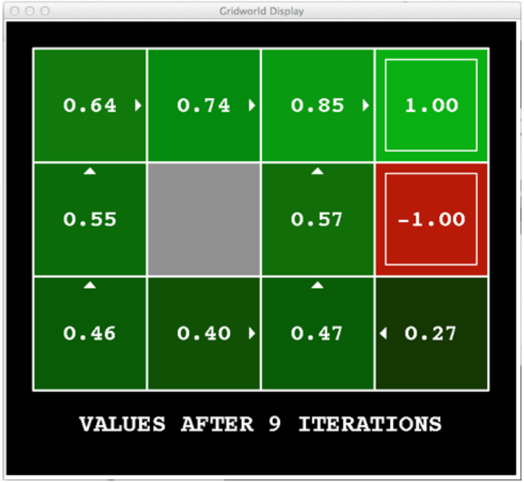
Example

k = 8

Noise = 0.2
Discount = 0.9
Living reward = 0

Example

k = 9



Noise = 0.2
Discount = 0.9
Living reward = 0

Example

k = 10

Noise = 0.2
Discount = 0.9
Living reward = 0

Example

k = 11

Noise = 0.2
Discount = 0.9
Living reward = 0

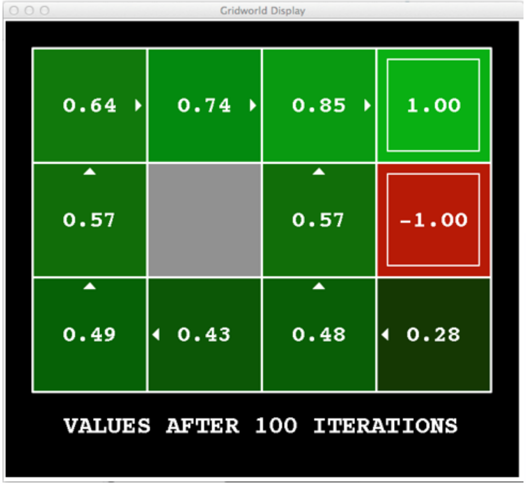
Example

k = 12

Noise = 0.2
Discount = 0.9
Living reward = 0

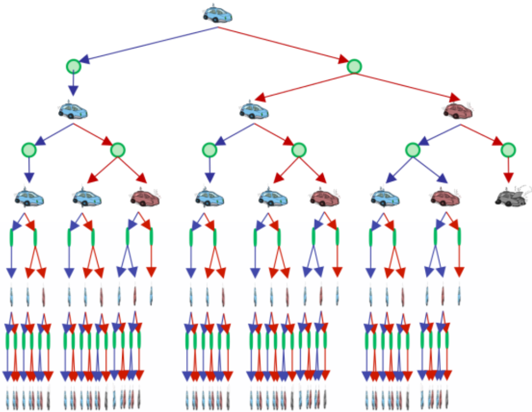
Example

k = 100

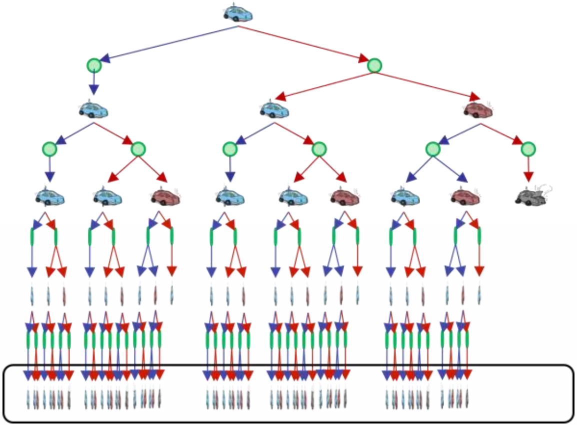


Noise = 0.2
Discount = 0.9
Living reward = 0

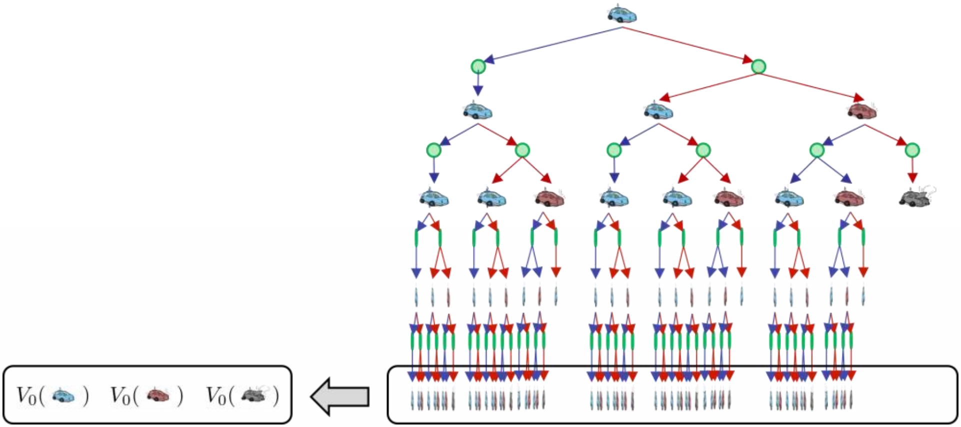
Computing Time-Limited Values



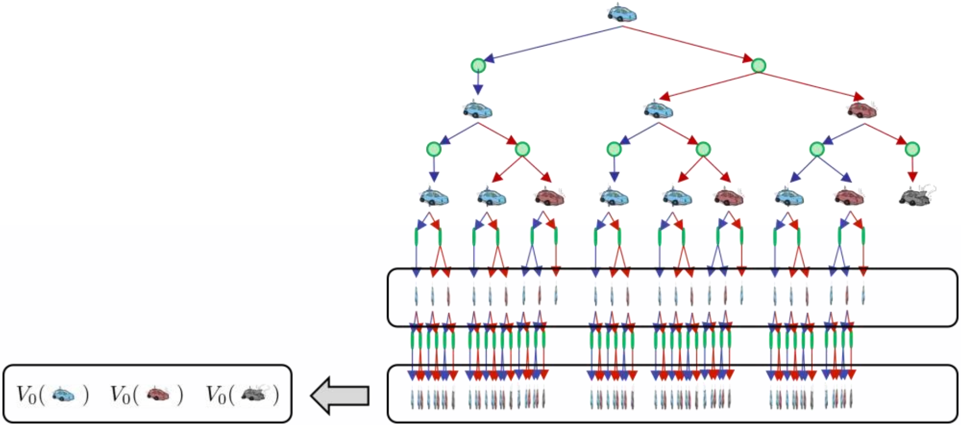
Computing Time-Limited Values



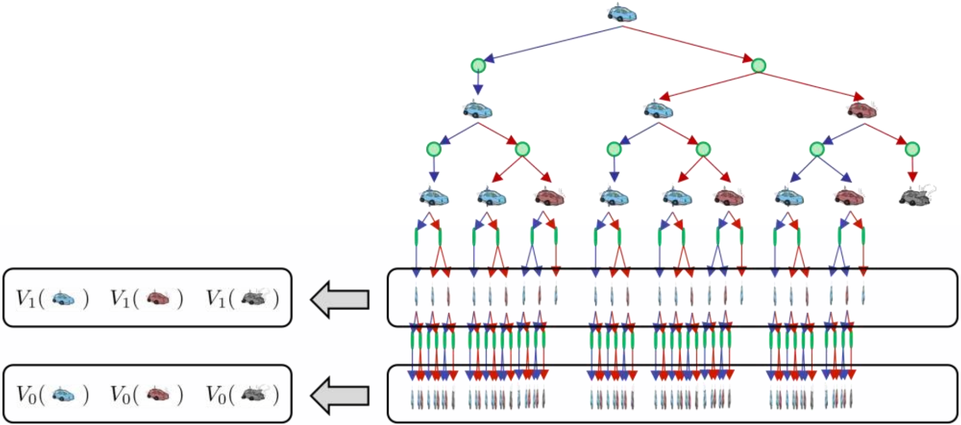
Computing Time-Limited Values



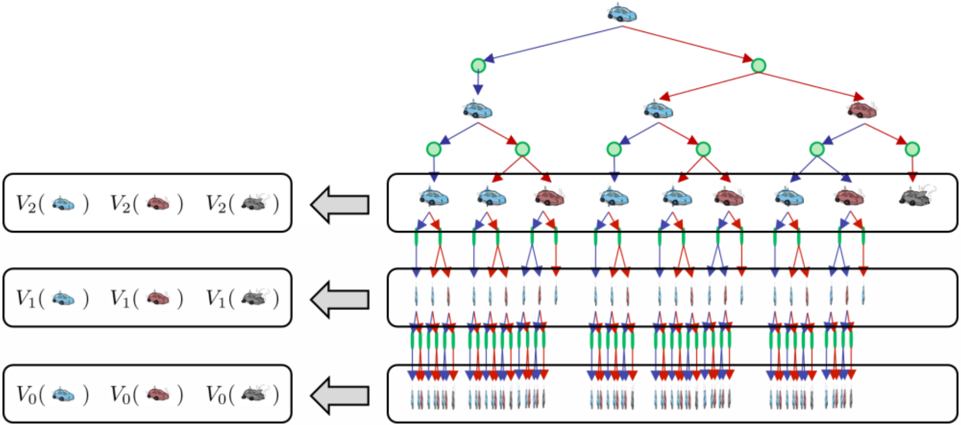
Computing Time-Limited Values



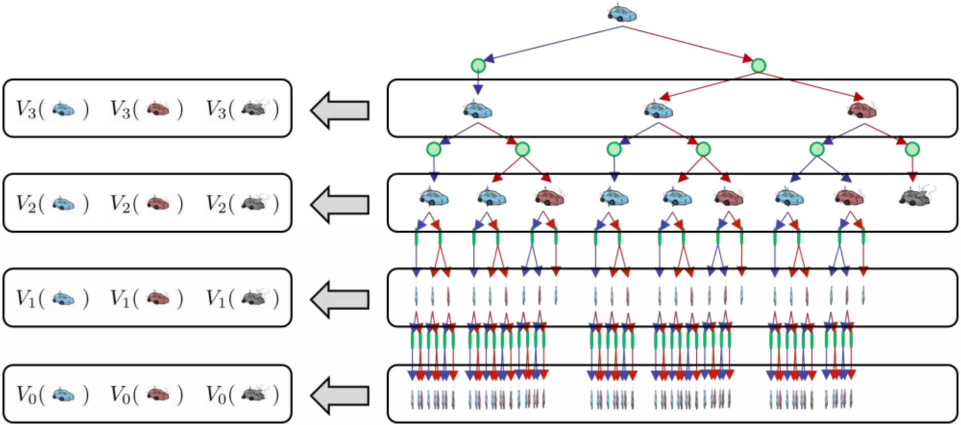
Computing Time-Limited Values



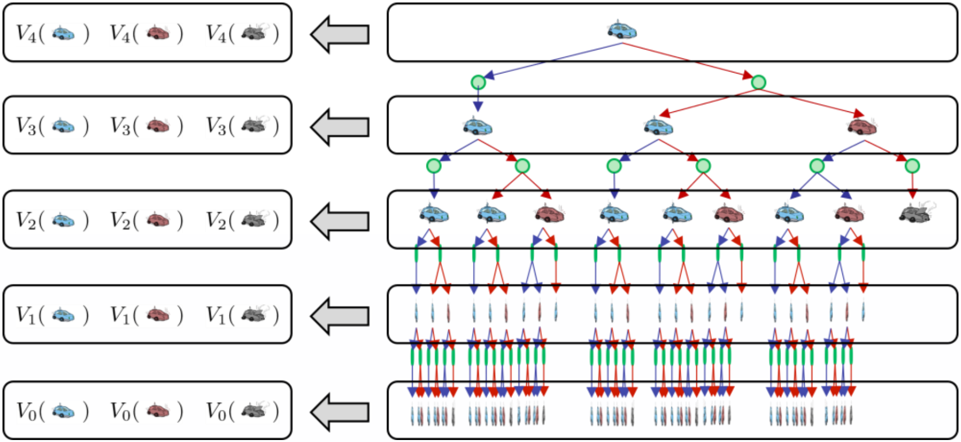
Computing Time-Limited Values



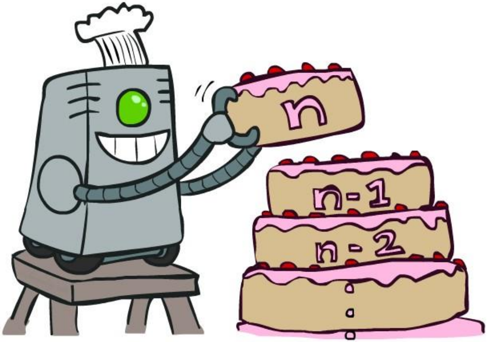
Computing Time-Limited Values



Computing Time-Limited Values



Value Iteration



Value Iteration

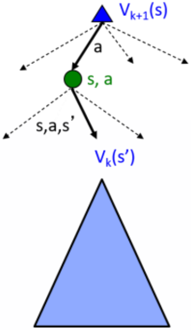
- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero

Value Iteration

- Start 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:

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



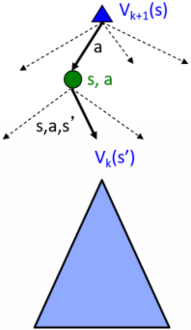
Value Iteration

- Start 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:

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

- Repeat until convergence



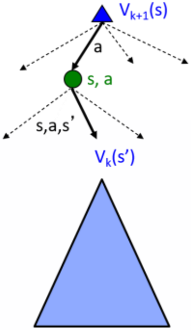
Value Iteration

- Start 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:

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

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



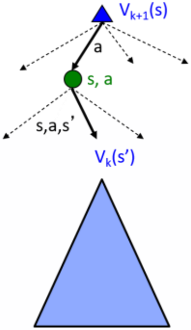
Value Iteration

- Start 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

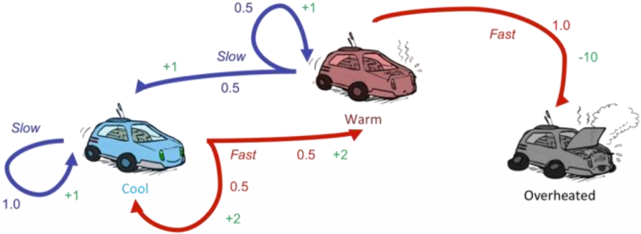


Example: Value Iteration

V_2

V_1

V_0

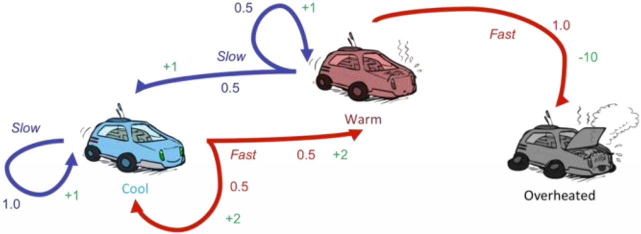


Assume no discount!

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

Example: Value Iteration

V_2			
V_1			
V_0	0	0	0

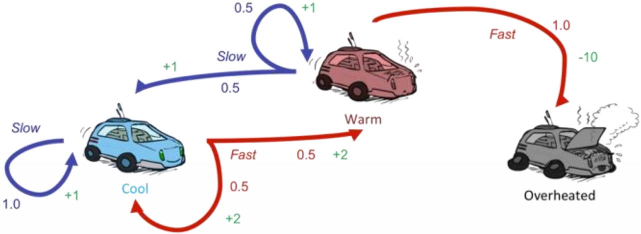


Assume no discount!

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

Example: Value Iteration

V_2			
V_1	2	1	0
V_0	0	0	0

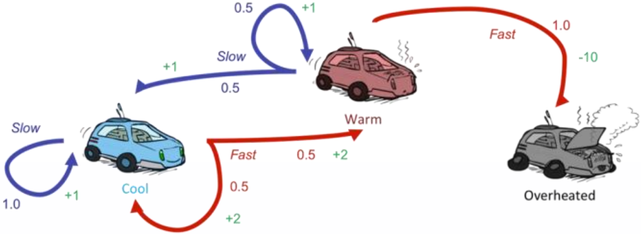


Assume no discount!

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

Example: Value Iteration

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



Assume no discount!

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

Suggested Reading

- Russell & Norvig: Chapter 17.1-17.3