Game-theoretic Foundations of Multi-agent Systems

Lecture 7: Stochastic Games

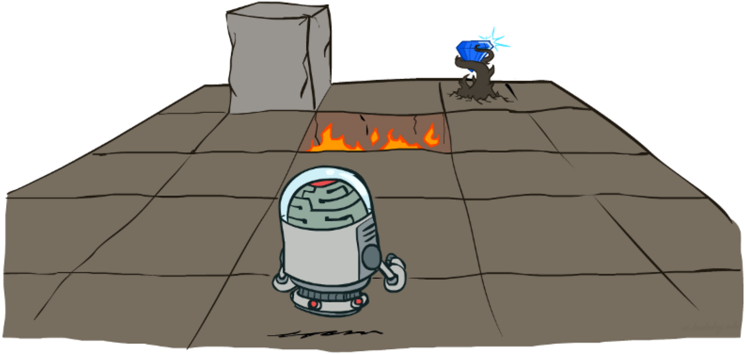
Seyed Majid Zahedi

UNIVERSITY OF WATERLOO

Outline

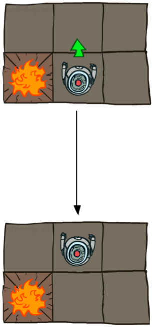
1. Markov Decision Processes
2. Definition of Stochastic Games
3. Strategies and Equilibria in Stochastic Games

Motivation: Non-deterministic Search in Grid World



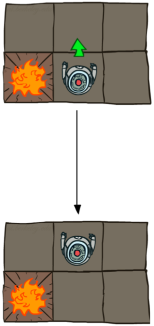
Grid World Actions

Deterministic

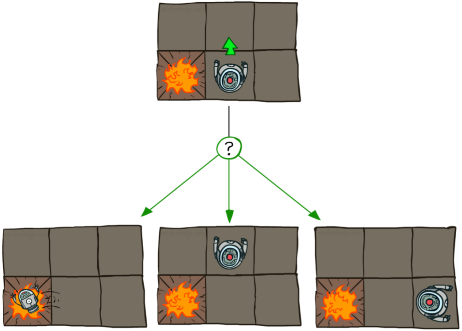


Grid World Actions

Deterministic

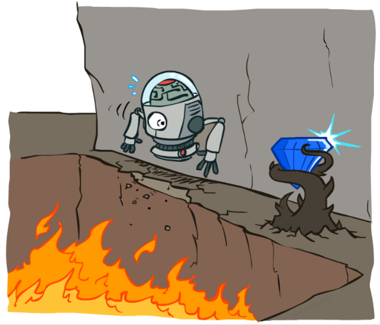


Stochastic



A Grid World Instance

- Agent lives in a grid
- Walls block agent's path
- Actions do not always go as planned
  - 80% of time, action "North" takes us north
  - 10% of time, "North" takes us west; 10% east
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Agent receives rewards at each step
- Goal is to maximize sum of rewards

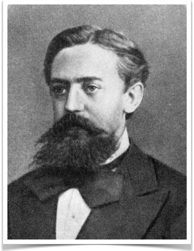


Change in Notation

- So far, we have used s to denote strategy profile
- In this lecture, we use π for strategy
- We use s to denote state

Markov Property

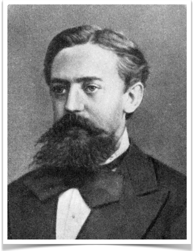
- Given present state, future and past are independent



Andrey Markov (1856-1922)

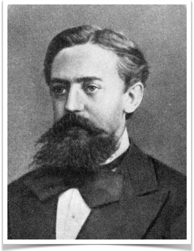
Markov Property

- Given present state, future and past are independent
- Future state depends only on current state and action



Andrey Markov (1856-1922)

Markov Property



Andrey Markov (1856-1922)

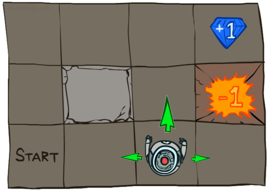
- Given present state, future and past are independent

- Future state depends only on current state and action

$$P(S_{t+1} = s \mid S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, \ldots, S_0 = s_0) =$$
$$P(S_{t+1} = s \mid S_t = s_t, A_t = a_t)$$

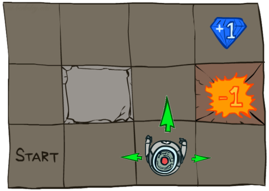
Markov Decision Processes: Formal Definition

- S is set of states and A is set of actions



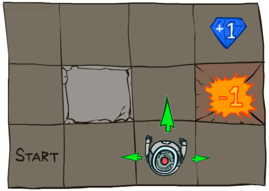
Markov Decision Processes: Formal Definition

- S is set of states and A is set of actions
- $p : S \times A \times S \mapsto [0, 1]$ specifies transition probabilities
  - $p(s, a, s')$ is probability of going to s' when taking a in s



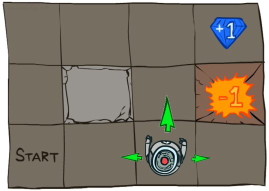
Markov Decision Processes: Formal Definition

- S is set of states and A is set of actions
- $p : S \times A \times S \mapsto [0, 1]$ specifies transition probabilities
  - $p(s, a, s')$ is probability of going to s' when taking a in s
- $r : S \times A \times S \mapsto \mathbb{R}$ returns reward
  - $r(s, a, s')$ is reward of going to s' when taking a in s



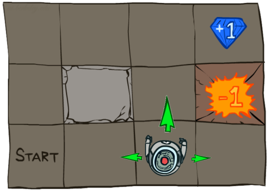
Markov Decision Processes: Formal Definition

- S is set of states and A is set of actions
- $p : S \times A \times S \mapsto [0, 1]$ specifies transition probabilities
  - $p(s, a, s')$ is probability of going to s' when taking a in s
- $r : S \times A \times S \mapsto \mathbb{R}$ returns reward
  - $r(s, a, s')$ is reward of going to s' when taking a in s
- There are two ways to aggregate rewards
  - Limit-average reward: $\lim_{T\to\infty} \sum_{t=1}^{T} r_t / T$
  - Future-discounted reward: $\sum_{t=1}^{\infty} \delta^{t-1} r_t$



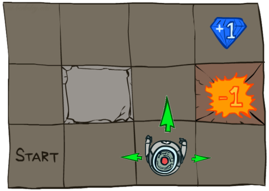
Markov Decision Processes: Formal Definition

- S is set of states and A is set of actions
- $p : S \times A \times S \mapsto [0,1]$ specifies transition probabilities
  - $p(s,a,s')$ is probability of going to s' when taking a in s
- $r : S \times A \times S \mapsto \mathbb{R}$ returns reward
  - $r(s,a,s')$ is reward of going to s' when taking a in s
- There are two ways to aggregate rewards
  - Limit-average reward: $\lim_{T \to \infty} \sum_{t=1}^{T} r_t / T$
  - Future-discounted reward: $\sum_{t=1}^{\infty} \delta^{t-1} r_t$



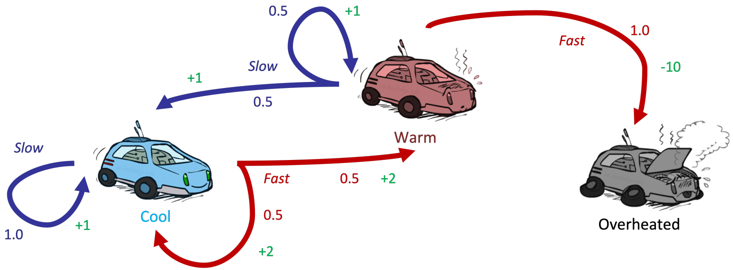
Markov Decision Processes: Formal Definition

- S is set of states and A is set of actions
- $p : S \times A \times S \mapsto [0, 1]$ specifies transition probabilities
  - $p(s, a, s')$ is probability of going to s' when taking a in s
- $r : S \times A \times S \mapsto \mathbb{R}$ returns reward
  - $r(s, a, s')$ is reward of going to s' when taking a in s
- There are two ways to aggregate rewards
  - Limit-average reward: $\lim_{T\to\infty} \sum_{t=1}^{T} r_t / T$
  - Future-discounted reward: $\sum_{t=1}^{\infty} \delta^{t-1} r_t$
    (we will focus on this)



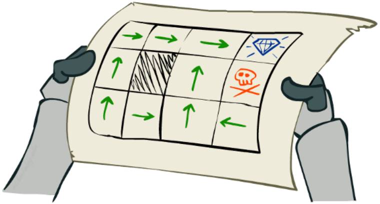
Example: Racing

- Autonomous car wants to travel far, quickly
- There are 3 states: Cool, Warm, Overheated, and 2 actions: Slow, Fast
- Going faster gets double reward



Policies

- A (stationary, deterministic) policy $\pi : S \mapsto A$ gives action for each state
- An optimal policy is one that maximizes expected utility if followed



Values of States

- Value function: $V^\pi(s)$ specifies value of following π starting in s

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \delta^{t-1} r_t(s_t, \pi(s_t), s_{t+1}) \mid s_0 = s\right]$$

Values of States

- Value function: $V^\pi(s)$ specifies value of following π starting in s

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \delta^{t-1} r_t(s_t, \pi(s_t), s_{t+1}) \mid s_0 = s\right]$$

- State-action value function: $Q^\pi(s, a)$ returns value of starting in s, taking a, and then continuing according to π

Values of States

- Value function: $V^\pi(s)$ specifies value of following π starting in s

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \delta^{t-1} r_t(s_t, \pi(s_t), s_{t+1}) \mid s_0 = s\right]$$

- State-action value function: $Q^\pi(s, a)$ returns value of starting in s, taking a, and then continuing according to π

- These two are related to each other by

$$Q^\pi(s, a) = \sum_{s'} p(s, a, s') \left(r(s, a, s') + \delta V^\pi(s')\right)$$

$$V^\pi(s) = Q^\pi(s, \pi(s))$$

Values of States

- Value function: $V^\pi(s)$ specifies value of following π starting in s

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \delta^{t-1} r_t(s_t, \pi(s_t), s_{t+1}) \mid s_0 = s\right]$$

- State-action value function: $Q^\pi(s, a)$ returns value of starting in s, taking a, and then continuing according to π

- These two are related to each other by

$$Q^\pi(s, a) = \sum_{s'} p(s, a, s') \left(r(s, a, s') + \delta V^\pi(s')\right)$$

$$V^\pi(s) = Q^\pi(s, \pi(s))$$

$$\Rightarrow \quad V^\pi(s) = \sum_{s'} p(s, \pi(s), s') \left(r(s, \pi(s), s') + \delta V^\pi(s')\right)$$

Policy Evaluation

Initialize $V_0^\pi(s) \leftarrow 0$ for all states s;
for $t = 1 \dots T$ **do**
  for *each state* s **do**
    $V_t^\pi(s) \leftarrow \sum_{s'} p(s, \pi(s), s') \left(r(s, \pi(s), s') + \delta V_{t-1}^\pi(s')\right)$

- How many iterations should we have (what should T be)?

Policy Evaluation

Initialize $V_0^\pi(s) \leftarrow 0$ for all states s;
for $t = 1 \ldots T$ **do**
  for *each state* s **do**
    $V_t^\pi(s) \leftarrow \sum_{s'} p(s, \pi(s), s') \left(r(s, \pi(s), s') + \delta V_{t-1}^\pi(s') \right)$

- How many iterations should we have (what should T be)?
- Repeat until values do not change much:

$$\max_{s \in S} |V_t^\pi(s) - V_{t-1}^\pi(s)| < \epsilon$$

Solving MDP: Bellman Equations

$$Q^*(s,a) = \sum_{s'} p(s,a,s')\left(r(s,a,s') + \delta V^*(s')\right)$$

$$V^*(s) = \max_{a \in A} \; Q^*(s,a)$$

$$\Rightarrow \; V^*(s) = \max_{a \in A} \; \sum_{s'} p(s,a,s')\left(r(s,a,s') + \delta V^*(s')\right)$$

Value Iteration

Initialize $V_0(s) \leftarrow 0$ for all states s;

repeat *until* $V(s)$ *converges for all* s

  for *each state* s **do**

    $V_t(s) \leftarrow \max_{a \in A} \sum_{s'} p(s,a,s')\left(r(s,a,s') + \delta V_{t-1}(s')\right)$

Value Iteration

Initialize $V_0(s) \leftarrow 0$ for all states s;
repeat *until* $V(s)$ *converges for all* s
    for *each state* s **do**
        $V_t(s) \leftarrow \max_{a \in A} \sum_{s'} p(s, a, s')\,(r(s, a, s') + \delta V_{t-1}(s'))$

- Bellman equations characterize the optimal values

Value Iteration

Initialize $V_0(s) \leftarrow 0$ for all states s;

repeat *until* $V(s)$ *converges for all* s

  for *each state* s **do**

    $V_t(s) \leftarrow \max_{a \in A} \sum_{s'} p(s,a,s')\left(r(s,a,s') + \delta V_{t-1}(s')\right)$

- Bellman equations characterize the optimal values
- Value iteration computes them

Value Iteration

Initialize $V_0(s) \leftarrow 0$ for all states s;
repeat *until $V(s)$ converges for all s*
  for *each state s* **do**
    $V_t(s) \leftarrow \max_{a \in A} \sum_{s'} p(s,a,s')\left(r(s,a,s') + \delta V_{t-1}(s')\right)$

- Bellman equations characterize the optimal values
- Value iteration computes them
- Value iteration is just a fixed-point solution method

Policy Extraction

- Given V^*, we can compute optimal policy as follows:

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} p(s,a,s') \left(r(s,a,s') + \delta V^*(s') \right)$$

Policy Extraction

- Given V^*, we can compute optimal policy as follows:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} p(s,a,s')\left(r(s,a,s') + \delta V^*(s')\right)$$

- Given Q^*, we can compute optimal policy as follows:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q^*(s,a)$$

Policy Extraction

- Given V^*, we can compute optimal policy as follows:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} p(s, a, s') \left(r(s, a, s') + \delta V^*(s') \right)$$

- Given Q^*, we can compute optimal policy as follows:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \; Q^*(s, a)$$

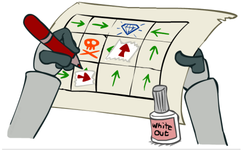
- Takeaway: actions are easier to select from Q-values than values!

Problems with Value Iteration

- It is slow - $O(S^2A)$
- The max at each state rarely changes
- The policy often converges long before the values

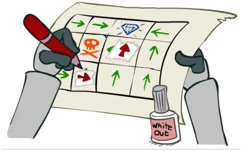
Policy Iteration

- (I) Policy evaluation: Calculate values for some fixed policy



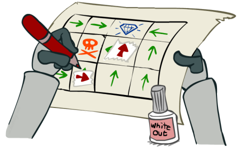
Policy Iteration

- (I) Policy evaluation: Calculate values for some fixed policy
- (II) Policy improvement: Extract policy given these values



Policy Iteration

- (I) Policy evaluation: Calculate values for some fixed policy
- (II) Policy improvement: Extract policy given these values
- Repeat steps until policy converges



Value Iteration vs Policy Iteration

- Both compute the same thing (all optimal values)

Value Iteration vs Policy Iteration

- Both compute the same thing (all optimal values)
- In value iteration:

Value Iteration vs Policy Iteration

- Both compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both values and (implicitly) policy

Value Iteration vs Policy Iteration

- Both compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both values and (implicitly) policy
  - We don't track policy, but taking max over actions implicitly recomputes it

Value Iteration vs Policy Iteration

- Both compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both values and (implicitly) policy
  - We don't track policy, but taking max over actions implicitly recomputes it
- In policy iteration:

Value Iteration vs Policy Iteration

- Both compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both values and (implicitly) policy
  - We don't track policy, but taking max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update values with fixed policy
    (each pass is fast because we consider only one action, not all of them)

Value Iteration vs Policy Iteration

- Both compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both values and (implicitly) policy
  - We don't track policy, but taking max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update values with fixed policy
    (each pass is fast because we consider only one action, not all of them)
  - After policy is evaluated, a new policy is chosen
    (slow like a value iteration pass)

Value Iteration vs Policy Iteration

- Both compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both values and (implicitly) policy
  - We don't track policy, but taking max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update values with fixed policy
    (each pass is fast because we consider only one action, not all of them)
  - After policy is evaluated, a new policy is chosen
    (slow like a value iteration pass)
  - New policy will be better (or we're done)

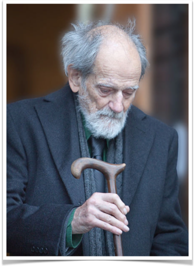
Value Iteration vs Policy Iteration

- Both compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both values and (implicitly) policy
  - We don't track policy, but taking max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update values with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - New policy will be better (or we're done)
- Both are dynamic programs for solving MDPs

Outline

Stochastic Games (a.k.a. Markov Games): Introduction

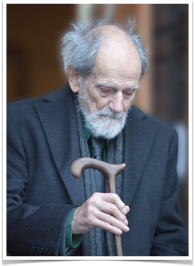
- Lloyd Shapley introduced stochastic games in early 1950s



Lloyd Shapley (1923-2016)

Stochastic Games (a.k.a. Markov Games): Introduction

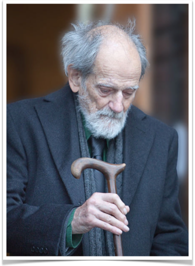
- Lloyd Shapley introduced stochastic games in early 1950s
- Stochastic games generalize repeated games



Lloyd Shapley (1923-2016)

Stochastic Games (a.k.a. Markov Games): Introduction

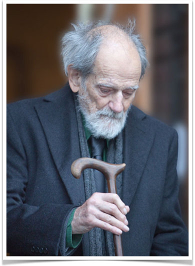
- Lloyd Shapley introduced stochastic games in early 1950s
- Stochastic games generalize repeated games
  - Agents repeatedly play games from set of stage games



Lloyd Shapley (1923-2016)

Stochastic Games (a.k.a. Markov Games): Introduction

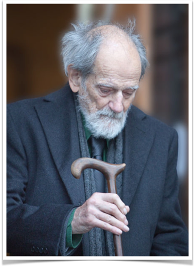
- Lloyd Shapley introduced stochastic games in early 1950s
- Stochastic games generalize repeated games
  - Agents repeatedly play games from set of stage games
- Stochastic games generalize Markov decision process



Lloyd Shapley (1923-2016)

Stochastic Games (a.k.a. Markov Games): Introduction

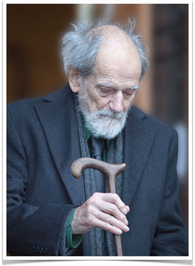
- Lloyd Shapley introduced stochastic games in early 1950s
- Stochastic games generalize repeated games
  - Agents repeatedly play games from set of stage games
- Stochastic games generalize Markov decision process
  - Game at each step only depends on outcome of previous step



Lloyd Shapley (1923-2016)

Stochastic Games (a.k.a. Markov Games): Introduction

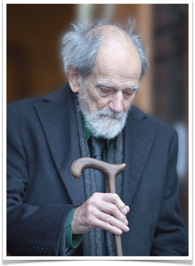
- Lloyd Shapley introduced stochastic games in early 1950s
- Stochastic games generalize repeated games
  - Agents repeatedly play games from set of stage games
- Stochastic games generalize Markov decision process
  - Game at each step only depends on outcome of previous step
- Single-state stochastic game = (infinitely) repeated game



Lloyd Shapley (1923-2016)

Stochastic Games (a.k.a. Markov Games): Introduction

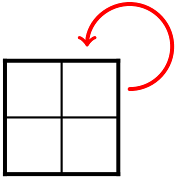
- Lloyd Shapley introduced stochastic games in early 1950s
- Stochastic games generalize repeated games
  - Agents repeatedly play games from set of stage games
- Stochastic games generalize Markov decision process
  - Game at each step only depends on outcome of previous step
- Single-state stochastic game = (infinitely) repeated game
- Single-agent stochastic game = MDP



Lloyd Shapley (1923-2016)

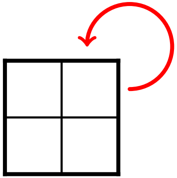
Repeated Games vs Stochastic Games

Repeated Games

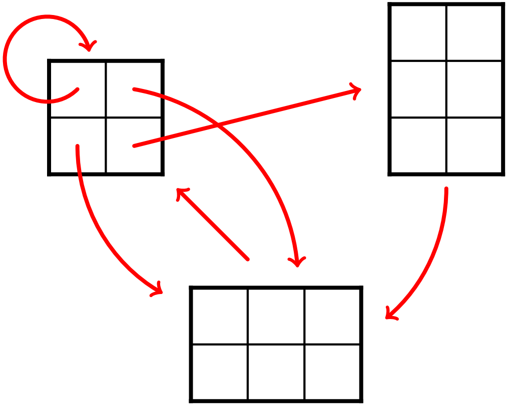


Repeated Games vs Stochastic Games

Repeated Games



Stochastic Games



Stochastic Games: Formal Definition[1]

- S is finite set of stage games
- N is finite set of n agents
- A_i is finite set of actions available to agent i
- $p : S \times A \times S \mapsto [0, 1]$ is transition probability function
  - $p(s, a, s')$ is probability of going from s to s' after action profile a
- $r_i : S \times A \mapsto \mathbb{R}$ is real-valued utility function for agent i
  - $r_i(s, a)$ is agent i's utility at state s for action profile a

[1] Note that this definition assume actions available to agents are the same across different stage games. Changing this assumption leads to a more involved notation.

Outline

1. Markov Decision Processes

2. Definition of Stochastic Games

3. Strategies and Equilibria in Stochastic Games

Stochastic Games: Strategies

- Let $h_t = (s_0, a_0, s_1, a_1, \ldots, a_{t-1}, s_t)$ denote history of t stages
- Let H_t be set of all possible histories of this length
- Set of all deterministic strategies for agent i is

$$\prod_{t, H_t} A_i$$

- Agents' strategies can consist of any mixture over deterministic strategies
- However, there are several restricted classes of strategies

Behavioral, Markov, and Stationary Strategies

- Behavioral strategy $\pi_i(h_t, a_i)$ returns probability of playing a_i for h_t
  - Mixing takes place at each history independently

Behavioral, Markov, and Stationary Strategies

- Behavioral strategy $\pi_i(h_t, a_i)$ returns probability of playing a_i for h_t
  - Mixing takes place at each history independently
- Markov strategy π_i is behavioral strategy s.t. $\pi_i(h_t, a_i) = \pi_i(h'_t, a_i)$ if $s_t = s'_t$
  - s_t and s'_t are final states of h_t and h'_t, respectively
  - For each t, distribution over actions depends only on current state

Behavioral, Markov, and Stationary Strategies

- Behavioral strategy $\pi_i(h_t, a_i)$ returns probability of playing a_i for h_t
  - Mixing takes place at each history independently
- Markov strategy π_i is behavioral strategy s.t. $\pi_i(h_t, a_i) = \pi_i(h'_t, a_i)$ if $s_t = s'_t$
  - s_t and s'_t are final states of h_t and h'_t, respectively
  - For each t, distribution over actions depends only on current state
- Stationary strategy π_i is a Markov strategy s.t. $\pi_i(h_{t_1}, a_i) = \pi_i(h'_{t_2}, a_i)$ if $s_{t_1} = s'_{t_2}$
  - s_{t_1} and s'_{t_2} are final states of h_{t_1} and h'_{t_2}, respectively
  - This removes possible dependence on time t

Markov-perfect Equilibrium (MPE)

- Strategy π is MPE if it is Markov strategy and is NE regardless of starting state

$$V_i^{\pi}(s) \geq V_i^{(\pi_i', \pi_{-i})}(s) \quad \forall i, s, \pi_i'$$

- MPE is similar to subgame-perfect equilibrium in perfect-information games

- Every n-player, general-sum, discounted-reward stochastic game has MPE

Computing Equilibrium

- Poly-time algorithms are not generally available for full class of stochastic games

Computing Equilibrium

- Poly-time algorithms are not generally available for full class of stochastic games
- However, they exist for several nontrivial sub-classes

Computing Equilibrium

- Poly-time algorithms are not generally available for full class of stochastic games
- However, they exist for several nontrivial sub-classes
- E.g., 2-player, general-sum, discounted-reward, single-controller stochastic games
  - Transitions depend on single agent: if $a_i = a'_i$, then $p(s, a, s') = p(s, a', s')$ $\forall s, s'$

Computing Equilibrium

- Poly-time algorithms are not generally available for full class of stochastic games
- However, they exist for several nontrivial sub-classes
- E.g., 2-player, general-sum, discounted-reward, single-controller stochastic games
  - Transitions depend on single agent: if $a_i = a_i'$, then $p(s, a, s') = p(s, a', s')$ $\forall s, s'$
- E.g., 2-player, general-sum, discounted-reward, separable-reward, state-independent-transition (SR-SIT) stochastic games
  - $r_i(s, a) = f(s) + g(a)$ $\forall i, s, a$, and
  - $p(s, a, s'') = p(s', a, s'')$ $\forall s, s', s'', a$

Computing Equilibrium

- Poly-time algorithms are not generally available for full class of stochastic games
- However, they exist for several nontrivial sub-classes
- E.g., 2-player, general-sum, discounted-reward, single-controller stochastic games
  - Transitions depend on single agent: if $a_i = a'_i$, then $p(s, a, s') = p(s, a', s')\ \forall s, s'$
- E.g., 2-player, general-sum, discounted-reward, separable-reward, state-independent-transition (SR-SIT) stochastic games
  - $r_i(s, a) = f(s) + g(a)\ \forall i, s, a$, and
  - $p(s, a, s'') = p(s', a, s'')\ \forall s, s', s'', a$
- E.g., 2-player, zero-sum, discounted-reward stochastic games

Shapley Algorithm: Finding MPE in 2-player Zero-sum Games

Initialize $V_0(s)$ arbitrarily for all s; ▷ Agent 1's utility for being in s

repeat *until* $V(s)$ *converges for all* s

  for *each state* s **do**

    Compute matrix game $G(s, V_{t-1})$:

    $u(s,a) = r(s,a) + \delta \sum_{s'} p(s,a,s') V_{t-1}(s')$

  for *each state* s **do**

    $V_t(s) \leftarrow \max_{\pi_1} \min_{\pi_2} u_1(s, \pi_1, \pi_2)$

Shapley Algorithm: Finding MPE in 2-player Zero-sum Games

Initialize $V_0(s)$ arbitrarily for all s; ▷ Agent 1's utility for being in s

repeat *until $V(s)$ converges for all s*

  for *each state s* **do**

    Compute matrix game $G(s, V_{t-1})$:

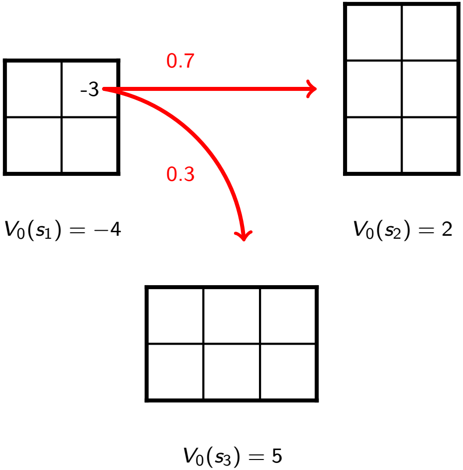
    $u(s,a) = r(s,a) + \delta \sum_{s'} p(s,a,s') V_{t-1}(s')$

  for *each state s* **do**

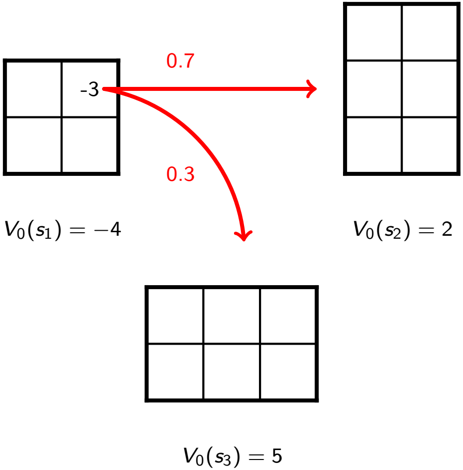
    $V_t(s) \leftarrow \max_{\pi_1} \min_{\pi_2} u_1(s, \pi_1, \pi_2)$

- Shapley's algorithm is extension of value iteration to stochastic games

Shapley Algorithm: Example



Shapley Algorithm: Example



$$-3 + \delta\,(0.7 \times 2 + 0.3 \times 5)$$
$$= -3 + 2.9\,\delta$$

	-3 + 2.9δ

$G(s_1, V_0)$

Pollatschek & Avi-Itzhak Algorithm (Extension of Policy Iteration)

Initialize $V(s)$ arbitrarily for all s; ▷ Agent 1's utility for being in s
repeat *until* $\pi_1(s)$ *and* $\pi_2(s)$ *converge for all* s
  for *each state* s **do**
    Compute matrix game $G(s, V)$ as in Shapley's algorithm;
    $\pi_1(s) \leftarrow$ maxmin strategy of Agent 1 in $G(s, V)$;
    $\pi_2(s) \leftarrow$ minmax strategy agents Agent 1 in $G(s, V)$;
  Calculate $V(s)$ with policy evaluation for π_1 and π_2

Acknowledgment

- This lecture is a slightly modified version of ones prepared by
  - Dan Klein and Pieter Abbeel [UC Berkeley CS 188]
  - Vincent Conitzer [Duke CPS 590.4]