25  **Sequential Problems**

This chapter extends simple games to a sequential context with multiple states. A *Markov game* (MG) can be viewed as a Markov decision process involving multiple agents with their own reward functions.\(^1\) In this formulation, transitions depend on the joint action and all agents seek to maximize their own reward. We generalize the response models and the Nash equilibrium solution concept from simple games to take into account the state transition model. The last part of this chapter discusses learning-based models where the agents adapt their policies based on information from observed interactions and knowledge of the reward and transition functions.

### 25.1 Markov Games

An MG (algorithm 25.1) extends a simple game to include a shared state \(s \in S\). The likelihood of transitioning from a state \(s\) to a state \(s'\) under a joint action \(a\) is given by the transition distribution \(T(s' | s, a)\). Each agent \(i\) receives a reward according to its own reward function \(R^i(s, a)\), which now additionally depends on the state. Example 25.1 sketches how traffic routing can be framed as a Markov game.

```plaintext
struct MG
  γ  # discount factor
  ℐ  # agents
  𝒮  # state space
  𝒜  # joint action space
  R  # joint reward function
  T  # transition function
end
```

1 Markov games, also called *stochastic games*, were originally studied in the 1950s around the same time as MDPs. L. S. Shapley, “Stochastic Games,” *Proceedings of the National Academy of Sciences*, vol. 39, no. 10, pp. 1095–1100, 1953. They were introduced into the multiagent artificial intelligence community decades later. M. L. Littman, “Markov Games as a Framework for Multi-Agent Reinforcement Learning,” in *International Conference on Machine Learning (ICML)*, 1994.
Consider commuters headed to work by car. Each car has a starting position and a destination. Each car can take any of several available roads to get to their destination, but these roads vary in the time it takes to drive them. The more cars that drive on a given road, the slower they all move.

This problem is a Markov game. The agents are the commuters in their cars, the states are the locations of all the cars on the roads, and the actions correspond to decisions of which road to take next. The state transition moves all car agents forward following their joint action. The negative reward is proportional to the time spent driving on a road.

The joint policy $\pi$ in a Markov game specifies a probability distribution over joint actions given the current state. As with Markov decision processes, we will focus on policies that depend on the current state rather than the past history because future states and rewards are conditionally independent of the history given the current state. In addition, we will focus on stationary policies, which do not depend on time. The probability that agent $i$ selects action $a$ at state $s$ is given by $\pi^i(a \mid s)$. We will often use $\pi(s)$ to represent a distribution over joint actions.

The utility of a joint policy $\pi$ from the perspective of agent $i$ can be computed using a variation of policy evaluation introduced in section 7.2 for MDPs. The reward to agent $i$ from state $s$ when following joint policy $\pi$ is

$$R^i(s, \pi(s)) = \sum_a R^i(s, a) \prod_{j \in I} \pi^j(a^j \mid s)$$  (25.1)

The probability of transitioning from state $s$ to $s'$ when following $\pi$ is

$$T(s' \mid s, \pi(s)) = \sum_a T(s' \mid s, a) \prod_{j \in I} \pi^j(a^j \mid s)$$  (25.2)

In an infinite horizon, discounted game, the utility for agent $i$ from state $s$ is

$$U^{\pi,i}(s) = R^i(s, \pi(s)) + \gamma \sum_{s'} T(s' \mid s, \pi(s)) U^{\pi,i}(s')$$  (25.3)

which can be solved exactly (algorithm 25.2).
25.2 Response Models

We can generalize the response models introduced in the previous chapter to Markov games. Doing so requires taking into account the state transition model.

25.2.1 Best Response

A response policy for agent $i$ is a policy $\pi^i$ that maximizes expected utility given the fixed policies of other agents $\pi^{-i}$. If the policies of the other agents are fixed, then the problem reduces to an MDP. This MDP has state space $S$ and action space $A^i$. We can define the transition and reward functions as follows:

$$T'(s' \mid s, a^i) = T(s' \mid s, a^i, \pi^{-i}(s)) \quad (25.4)$$

$$R'(s, a^i) = R^i(s, a^i, \pi^{-i}(s)) \quad (25.5)$$

Because this is a best response for agent $i$, the MDP only uses reward $R^i$. Solving this MDP results in a best response policy for agent $i$. Algorithm 25.3 provides an implementation.
function best_response(\(\mathcal{P}:\mathcal{MG}, \pi, i\))
    \(S, A, R, T, \gamma = \mathcal{P}.S, \mathcal{P}.A, \mathcal{P}.R, \mathcal{P}.T, \mathcal{P}.\gamma\)
    \(T'(s, ai, s') = \text{transition}(\mathcal{P}, s, \text{joint}(\pi, \text{SimpleGamePolicy}(ai), i), s')\)
    \(R'(s, ai) = \text{reward}(\mathcal{P}, s, \text{joint}(\pi, \text{SimpleGamePolicy}(ai), i), i)\)
    \(\pi_i = \text{solve}(\text{MDP}(\gamma, S, A[i], T', R'))\)
    return MGPolicy(s \Rightarrow \text{SimpleGamePolicy}(\pi_i(s)) \text{ for } s \text{ in } S)
end

25.2.2 Softmax Response

Similar to what was done in the previous chapter, we can define a softmax response policy, which assigns a stochastic response to the policies of the other agents at each state. As we did in the construction of a deterministic best response policy, we solve an MDP where the agents with the fixed policies \(\pi^{-i}\) are folded into the environment. We then extract the action value function \(Q(s, a)\) using one-step lookahead. The softmax response is then

\[
\pi^i(a^i | s) \propto \exp(\lambda Q(s, a^i)) \tag{25.6}
\]

with precision parameter \(\lambda \geq 0\). Algorithm 25.4 provides an implementation. This approach can be used to generate hierarchical softmax solutions (section 24.7). In fact, we can use algorithm 24.9 directly.

function softmax_response(\(\mathcal{P}:\mathcal{MG}, \pi, i, \lambda\))
    \(S, A, R, T, \gamma = \mathcal{P}.S, \mathcal{P}.A, \mathcal{P}.R, \mathcal{P}.T, \mathcal{P}.\gamma\)
    \(T'(s, ai, s') = \text{transition}(\mathcal{P}, s, \text{joint}(\pi, \text{SimpleGamePolicy}(ai), i), s')\)
    \(R'(s, ai) = \text{reward}(\mathcal{P}, s, \text{joint}(\pi, \text{SimpleGamePolicy}(ai), i), i)\)
    \(\text{mdp} = \text{MDP}(\gamma, S, \text{joint}(A), T', R')\)
    \(\pi_i = \text{solve}(\text{mdp})\)
    \(Q(s, a) = \text{lookahead}(\text{mdp}, \pi_i.U, s, a)\)
    \(p(s) = \text{SimpleGamePolicy}(a \Rightarrow \exp(\lambda*Q(s, a)) \text{ for } a \text{ in } A[i])\)
    return MGPolicy(s \Rightarrow p(s) \text{ for } s \text{ in } S)
end

25.3 Nash Equilibrium

The Nash equilibrium concept can be generalized to Markov games. As with simple games, all agents perform a best response to one another, and have no incentive to deviate. All finite Markov games with a discounted infinite horizon have a Nash equilibrium.3

Because we assume policies are stationary, in that they do not vary over time, the Nash equilibria covered here are stationary Markov perfect equilibria.

We can find a Nash equilibrium by solving a nonlinear optimization problem similar to the one we solved in the context of simple games. This problem minimizes the sum of the lookahead utility deviations and constrains the policies to be valid distributions:

\[
\begin{align*}
\text{minimize} & \quad \sum_{i \in I} \sum_{s} \left( U^i(s) - Q^i(s, \pi(s)) \right) \\
\text{subject to} & \quad U^i(s) \geq Q^i(s, a^i, \pi^{-i}(s)) \quad \text{for all } i, s, a^i \\
& \quad \sum_{a^i} \pi^i(a^i \mid s) = 1 \quad \text{for all } i, s \\
& \quad \pi^i(a^i \mid s) \geq 0 \quad \text{for all } i, s, a^i
\end{align*}
\]  \hspace{1cm} (25.7)

where

\[
Q^i(s, \pi(s)) = R^i(s, \pi(s)) + \gamma \sum_{s'} T(s' \mid s, \pi(s)) U^i(s')
\]  \hspace{1cm} (25.8)

This nonlinear optimization problem is implemented and solved in algorithm 25.5.4

25.4 Fictitious Play

As we did in the context of simple games, we can take a learning-based approach to arrive at joint policies by running agents in simulation. Algorithm 25.6 generalizes the simulation loop introduced in the previous chapter to handle state transitions. The various policies run in simulation update themselves based on the state transitions and the actions taken by the various agents.

One approach for updating policies is to use a generalization of fictitious play (algorithm 25.7) from the previous chapter.5 It involves maintaining a maximum-likelihood model over the policies of the other agents. The maximum likelihood model tracks the state in addition to the action being taken by each agent. We track the number of times agent \( j \) takes action \( a^j \) in state \( s \), storing it in a table \( N(j, a^j, s) \), typically initialized to 1. Then, we can compute the best response assuming that each agent \( j \) follows the state-dependent stochastic policy:

\[
\pi^j(a^j \mid s) \propto N^j(j, a^j, s)
\]  \hspace{1cm} (25.9)

After observing joint action \( a \) in states \( s \), we update

\[
N(j, a^j, s) \leftarrow N(j, a^j, s) + 1
\]  \hspace{1cm} (25.10)

for each agent \( j \).
Algorithm 25.5. This nonlinear program computes a Nash equilibrium for a Markov game \( \mathcal{P} \).

```plaintext
function tensorform(\( \mathcal{P} \)::MG)
    \( \mathcal{T} \), \( S \), \( A \), \( R \), \( T \) = tensorform(\( \mathcal{P} \))
    \( \mathcal{T}' \) = eachindex(\( \mathcal{T} \))
    \( S' \) = eachindex(\( S \))
    \( A' \) = [eachindex(\( A[i] \)) for \( i \) in \( \mathcal{T} \)]
    \( R' \) = [\( R(s,a) \) for \( s \) in \( S \), \( a \) in joint(\( A \))] \( \mathcal{S}' \) = [\( T(s,a,s') \) for \( s \) in \( S \), \( a \) in joint(\( A \)), \( s' \) in \( S \)]
    return \( \mathcal{T}' \), \( S' \), \( A' \), \( R' \), \( T' \)
end

function solve!(\( M \)::NashEquilibrium, \( \mathcal{P} \)::MG)
    \( \mathcal{T} \), \( S \), \( A \), \( R \), \( T \) = tensorform(\( \mathcal{P} \))
    \( S' \), \( A' \), \( \gamma \) = \( \mathcal{P} \).
    model = Model(Ipopt.Optimizer)
    @variable(model, \( U[i,s] \) for \( i \) in \( \mathcal{T} \), \( s \) in \( S \), \( ai=\mathcal{A}[i] \) ≥ 0)
    @NLobjective(model, Min, sum(\( U[i,s] \) - sum(prod(\( \mathcal{A}'[i][j] \) for \( j \) in \( \mathcal{T} \) ) ) * (\( R[s,y][i] \) + \( \gamma \) * sum(T[s,y,s']*U[i,s'] for \( s' \) in \( S \) )) for \( y,a \) in enumerate(joint(\( A \))) ) for \( i \) in \( T \), \( s \) in \( S \))
    @NLconstraint(model, \( i=\mathcal{T} \), \( s=\mathcal{S} \), \( ai=\mathcal{A}[i] \), \( U[i,s] \) ≥ sum( prod(\( j=\mathcal{T} \) ? \( a[j] \) : \( 1.0 \) : \( 0.0 \) ) : \( \mathcal{A} \) for \( j \) in \( \mathcal{T} \) ) * (\( R[s,y][i] \) + \( \gamma \) * sum(T[s,y,s']*U[i,s'] for \( s' \) in \( S \) )) for \( y,a \) in enumerate(joint(\( A \))) )
    @constraint(model, \( i=\mathcal{T} \), \( s=\mathcal{S} \), sum(\( \mathcal{A}'[i][s] \) for \( ai \) in \( \mathcal{A}[i] \) ) == 1)  optimize!(model)
    \( \pi' \) = value.(\( \pi \))
    \( \pi'_i(s) = SimpleGamePolicy(\( \mathcal{A}'[i][ai] \Rightarrow \pi'[i,s,ai] \) for \( ai \) in \( \mathcal{A}[i] \) ) \( \pi'_i(s) = MGPolicy(\mathcal{S}'[s] \Rightarrow \pi'_i(s) \) for \( s \) in \( S \))
    return [\( \pi'_i(s) \) for \( i \) in \( \mathcal{T} \)]
end
```
As the distributions of the other agents’ actions change, we must update the utilities. The utilities in Markov games are significantly more difficult to compute compared to simple games because of the state dependency. As described in section 25.2.1, any assignment of fixed policies of others $\pi^{-i}$ determined from equation (25.9) induces an MDP. Instead of solving an MDP at each update, it is common to apply the update periodically, a strategy adopted from asynchronous value iteration. An example of fictitious play is given in example 25.2.

Our policy $\pi^i(s)$ for a state $s$ is derived from a given opponent model $\pi^{-i}$ and computed utility $U^i$. We then select a best response:

$$\arg\max_a Q^i(s, a, \pi^{-i})$$

(25.11)

In the implementation here, we use the property that each state of a Markov game policy is a simple game policy whose reward is the corresponding $Q^i$.

### 25.5 Gradient Ascent

We can use gradient ascent (algorithm 25.8) to learn policies in a way similar to what was done in the previous chapter for simple games. The state must now be considered and requires learning the action value function. At each time step $t$, all agents perform joint actions $a_t$ in a state $s_t$. As in gradient ascent for simple
Algorithm 25.7. Fictitious play for agent $i$ in an MG $\mathcal{P}$ that maintains counts $\mathbf{N}_i$ of other agent action selections over time for each state and averages them assuming this is their stochastic policy. It then computes a best response to this policy and performs the corresponding utility-maximizing action.

```plaintext
mutable struct MGFictitiousPlay
    $\mathcal{P}$ # Markov game
    $i$ # agent index
    $Q_i$ # state-action value estimates
    $\mathbf{N}_i$ # state-action counts
end

function MGFictitiousPlay($\mathcal{P}$::MG, $i$)
    $T$, $S$, $A$, $R =$ $\mathcal{P}$.T, $\mathcal{P}$.S, $\mathcal{P}$.A, $\mathcal{P}$.R
    $Q_i = \text{Dict}((s, a) \mapsto R(s, a)[i] \text{ for } s \text{ in } S \text{ for } a \text{ in } \text{joint}(A))$
    $\mathbf{N}_i = \text{Dict}((j, s, a) \mapsto 1.0 \text{ for } j \text{ in } T \text{ for } s \text{ in } S \text{ for } aj \text{ in } A[j])$
    return MGFictitiousPlay($\mathcal{P}$, $i$, $Q_i$, $\mathbf{N}_i$)
end

function ($\pi_i$::MGFictitiousPlay)($s$)
    $T$, $i$, $Q_i = \pi_i.T$, $\pi_i.i$, $\pi_i.Q_i$
    $T$, $S$, $A$, $T$, $R$, $\gamma = \mathcal{P}$.T, $\mathcal{P}$.S, $\mathcal{P}$.A, $\mathcal{P}$.T, $\mathcal{P}$.R, $\mathcal{P}.\gamma$
    $\pi_i'(i, s) = \text{SimpleGamePolicy}(ai \mapsto \pi_i.N_i[i, s, ai] \text{ for } ai \text{ in } A[i])$
    $\pi_i'(i) = \text{MGPolicy}(s \mapsto \pi_i'(i, s) \text{ for } s \text{ in } S)$
    $\pi = [\pi_i'(i) \text{ for } i \text{ in } T]$
    $U(s, \pi) = \text{sum}(\pi_i.Q_i[s, a] \times \text{probability}($$\mathcal{P}, s, \pi, a$) \text{ for } a \text{ in } \text{joint}(A))$
    $Q(s, \pi) = \text{reward}($$\mathcal{P}, s, \pi, i$) + $\gamma \times \text{sum}($$\text{transition}($$\mathcal{P}, s, \pi, s', \pi)$ $\times U(s', \pi)$ $\text{for } s' \text{ in } S)$
    $Q(ai) = Q(s, \text{joint}(\pi, \text{SimpleGamePolicy}(ai), i))$
    $ai = \text{argmax}(Q, \mathcal{P}.$$\mathcal{A}[\pi_i.i])$
    return SimpleGamePolicy(ai)
end

function update!(($\pi_i$::MGFictitiousPlay), $s$, $a$, $s'$)
    $T$, $i$, $Q_i = \pi_i.T$, $\pi_i.i$, $\pi_i.Q_i$
    $T$, $S$, $A$, $T$, $R$, $\gamma = \mathcal{P}$.T, $\mathcal{P}$.S, $\mathcal{P}$.A, $\mathcal{P}$.T, $\mathcal{P}$.R, $\mathcal{P}.\gamma$
    for ($j, aj$) in enumerate($a$)
        $\pi_i.N_i[j, s, aj] += 1$
    end
    $\pi_i'(i, s) = \text{SimpleGamePolicy}(ai \mapsto \pi_i.N_i[i, s, ai] \text{ for } ai \text{ in } A[i])$
    $\pi_i'(i) = \text{MGPolicy}(s \mapsto \pi_i'(i, s) \text{ for } s \text{ in } S)$
    $\pi = [\pi_i'(i) \text{ for } i \text{ in } T]$
    $U(\pi, s) = \text{sum}(\pi_i.Q_i[s, a] \times \text{probability}($$\mathcal{P}, s, \pi, a$) \text{ for } a \text{ in } \text{joint}(A))$
    $Q(s, a) = $ $R(s, a)[i] + \gamma \times \text{sum}(T(s, a, s', \pi) \times U(\pi, s') \text{ for } s' \text{ in } S)$
    for $a$ in $\text{joint}(A)$
        $\pi_i.Q_i[s, a] = Q(s, a)$
    end
end
```

© 2022 Massachusetts Institute of Technology, shared under a under a Creative Commons CC-BY-NC-ND license.
2021-07-20 17:04:22-07:00, revision 2babf1f, comments to bugs@algorithmsbook.com
The predator-prey hex world Markov game (appendix F.13) has one predator (red) and one prey (blue). If the predator catches the prey, it receives a reward of 10 and the prey receives a reward of $-100$. Otherwise, both agents receive a $-1$ reward. The agents move simultaneously. We apply fictitious play with resets to the initial state every 10 steps.

We observe that the predator learns to chase the prey and the prey learns to flee. Interestingly, the predator also learns that the prey runs to the right corner and waits. Here, the prey learns that if it waits at this corner, it can flee from the predator immediately as it jumps toward the prey. In this case, it escapes the predator and runs to the other side of the map.

Below is a plot of the learned opponent model of the highlighted state (both predator and prey hex locations) for both the predator and the prey.
games, an agent $i$ assumes the agents’ policies $\pi_i^{-i}$ are the observed actions $a_t^{-i}$.

The gradient is then

$$\frac{\partial U^{\pi_i^t(s_t)}}{\partial \pi_i^t(a_i^t | s_t)} = \frac{\partial}{\partial \pi_i^t(a_i^t | s_t)} \left( \sum_a \prod_j \pi_j^t(a_j^t | s_t) Q^{\pi_j^t(s_t, a_t)} \right)$$

$$= Q^{\pi_i^t(s_t, a_i^t, a_t^{-i})}$$

(25.12)

(25.13)

The gradient step follows a similar pattern as in the previous chapter, except the state $s$ is included and the expected utility estimate $Q_i^t$ is used:

$$\pi_{i+1}^t(a_i^t | s_t) = \pi_i^t(a_i^t | s_t) + \alpha_t Q^i(s_t, a_i^t, a_t^{-i})$$

(25.14)

Again, this update may require projection to ensure a valid probability distribution.

As with fictitious play in the previous section, we must estimate $Q_i^t$. We can use Q-learning:

$$Q_{i+1}^t(s_t, a_t) = Q_i^t(s_t, a_t) + \alpha_t \left( R_i^t(s_t, a_t) + \gamma \max_{a_i'} Q_i^t(s_{t+1}, a_i', a_t^{-i}) - Q_i^t(s_t, a_t) \right)$$

(25.15)

We can use the inverse square-root learning rate $\alpha_t = 1/\sqrt{t}$. Exploration is also necessary. We can use an $\epsilon$-greedy strategy, perhaps also with $\epsilon_t = 1/\sqrt{t}$.

### 25.6 Nash Q-Learning

Another learning-based approach is Nash Q-learning (algorithm 25.9), which borrows inspiration from Q-learning (section 17.2). The method maintains an estimate of the action value function, which is adapted as the agents react to each other’s changing policies. In the process of updating the action value function, it computes a Nash equilibrium to model the behavior of the other agents.

An agent following Nash Q-learning maintains an estimate of a joint action value function $Q(s, a)$. This action value function is updated after every state transition using a Nash equilibrium computed from a simple game constructed from this value function. After a transition from $s$ to $s'$ following the joint action $a$, we construct a simple game with the same number of agents and the same joint action space, but have the reward function equal to the estimated value of $s'$ such that $R(a') = Q(s', a')$. The agent computes a Nash equilibrium policy $\pi'$ over the...