24 Multiagent Reasoning

So far, we have focused on making rational decisions for a single agent. These models have natural extensions to multiple agents. New challenges emerge as agents interact; agents can aid each other or act in their own best interests. Multiagent reasoning is a subject of *game theory*.[1] This chapter builds upon the concepts introduced earlier, extending them to multiagent contexts. We will discuss the foundational game theoretic approaches to compute decision strategies and multiagent equilibria.

24.1 Simple Games

A *simple game* (algorithm 24.1) is a fundamental model for multiagent reasoning.[2] Each agent $i \in \mathcal{I}$ selects an action a^i to maximize their own accumulation of reward r^i. The *joint action space* $\mathcal{A} = \mathcal{A}^1 \times \cdots \times \mathcal{A}^k$ consists of all possible permutations of the actions $\mathcal{A}^i$ available to each agent. The actions selected simultaneously across agents can be combined to form a *joint action* $\mathbf{a} = (a^1, \ldots, a^k)$ from this joint action space.[3] The *joint reward function* $\mathbf{R}(\mathbf{a}) = (R^1(\mathbf{a}), \ldots, R^k(\mathbf{a}))$ represents the reward produced by the joint action $\mathbf{a}$. The *joint reward* is written $\mathbf{r} = (r^1, \ldots, r^k)$. Simple games do not include states or transition functions. Example 24.1 introduces a simple game.

```
struct SimpleGame
    γ  # discount factor
    𝓘  # agents
    𝒜  # joint action space
    R  # joint reward function
end
```

[1] Game theory is a broad field. Several standard introductory books include: D. Fudenberg and J. Tirole, *Game Theory*. MIT Press, 1991. R. B. Myerson, *Game Theory: Analysis of Conflict*. Harvard University Press, 1997. Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[2] Simple games encompass *normal form games* (also called *standard form games* or *matrix games*), finite horizon *repeated games*, and infinite horizon discounted repeated games. Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[3] A joint action is also called an *action profile*.

Algorithm 24.1. Data structure for a simple game.

The prisoner's dilemma is a two-agent, two-action simple game involving two prisoners that are on trial. They can choose to *cooperate* and remain silent about their shared crime, or *defect* and blame the other for their crime. If they both cooperate, they both serve time for one year. If agent i cooperates and the other agent defects, then i serves no time and the other serves four years. If both defect, then they both serve three years.

Two-agent simple games can be represented by a table. Rows represent actions for agent 1. Columns represent actions for agent 2. The rewards for agent 1 and 2 are shown in each cell.

Example 24.1. A simple game known as the prisoner's dilemma. Additional detail is provided in appendix F.10.

agent 1 \ agent 2	cooperate	defect
cooperate	$-1, -1$	$-4, 0$
defect	$0, -4$	$-3, -3$

A *joint policy* $\boldsymbol{\pi}$ specifies a probability distribution over joint actions taken by the agents. Joint policies can be decomposed into individual agent policies. The probability agent i selects action a is given by $\pi^i(a)$. In game theory, a deterministic policy is called a *pure strategy* and a stochastic policy is called a *mixed strategy*. The utility of a joint policy $\boldsymbol{\pi}$ from the perspective of agent i is

$$U^i(\boldsymbol{\pi}) = \sum_{\mathbf{a} \in \mathcal{A}} R^i(\mathbf{a}) \prod_{j \in \mathcal{I}} \pi^j(a^j) \tag{24.1}$$

Algorithm 24.2 implements routines for representing policies and computing their utility.

A *zero-sum game* is a type of simple game where the sum of rewards across agents is zero. Here, any gain of an agent results at a loss to the other agents. A zero-sum game with two agents $\mathcal{I} = \{1, 2\}$ has opposing reward functions $R^1(\mathbf{a}) = -R^2(\mathbf{a})$. They are typically solved with algorithms specialized for this reward structure. Example 24.2 describes such a game.

24.2 Response Models

Before discussing different concepts for solving for a joint policy, we will begin by discussing how to model the *response* of a single agent i given fixed policies of the other agents. We will use the notation $-i$ as shorthand for $(1, \ldots, i-1, i+1, \ldots, k)$. Using this notation, a joint action is written $\mathbf{a} = (a^i, \mathbf{a}^{-i})$, a joint reward is written $\mathbf{R}(a^i, \mathbf{a}^{-i})$, and a joint policy $\boldsymbol{\pi} = (\pi^i, \boldsymbol{\pi}^{-i})$. This section discusses different approaches for computing a response to a known $\boldsymbol{\pi}^{-i}$.

Example 24.2. The well-known game of rock-paper-scissors is an example of a zero-sum game. Appendix F.11 provides additional detail.

Rock-paper-scissors is a zero-sum game for two agents. Each agent selects either rock, paper, or scissors. Rock wins against scissors, paper wins against rock, and scissors wins against paper, with a reward of 1 for the winner and -1 from the loser. If the agents select the same action, both receive 0 reward. Generally, two-agent repeated games can be represented as a sequence of payoff matrices, as shown below.

$t = 1$

agent 1 \ agent 2	rock	paper	scissors
rock	$0, 0$	$-1, 1$	$1, -1$
paper	$1, -1$	$0, 0$	$-1, 1$
scissors	$-1, 1$	$1, -1$	$0, 0$

$t = 2$

agent 1 \ agent 2	rock	paper	scissors
rock	$0, 0$	$-1, 1$	$1, -1$
paper	$1, -1$	$0, 0$	$-1, 1$
scissors	$-1, 1$	$1, -1$	$0, 0$

$\cdots$

$\cdots$

24.2.1 Best Response

A *best response* of agent i to the policies of the other agents $\boldsymbol{\pi}^{-i}$ is a policy π^i that satisfies

$$U^i(\pi^i, \boldsymbol{\pi}^{-i}) \geq U^i(\pi^{i\prime}, \boldsymbol{\pi}^{-i}) \tag{24.2}$$

for all other policies $\pi^{i\prime} \neq \pi^i$. In other words, a best response for an agent is a policy where there is no incentive for them to change their policy given a fixed set of policies of the other agents. There may be multiple such best responses.

If we restrict ourselves to deterministic policies, a *deterministic best response* to opponent policies $\boldsymbol{\pi}^{-i}$ is straightforward to compute. We simply iterate over all of the actions of agent i and return the one that maximizes utility:

$$\arg\max_{a^i \in \mathcal{A}^i} U^i(a^i, \boldsymbol{\pi}^{-i}) \tag{24.3}$$

Algorithm 24.3 provides an implementation.

```
struct SimpleGamePolicy
    p # dictionary mapping actions to probabilities

    function SimpleGamePolicy(p::Base.Generator)
        return SimpleGamePolicy(Dict(p))
    end

    function SimpleGamePolicy(p::Dict)
        vs = collect(values(p))
        vs ./= sum(vs)
        return new(Dict(k ⇒ v for (k,v) in zip(keys(p), vs)))
    end

    SimpleGamePolicy(ai) = new(Dict(ai ⇒ 1.0))
end

(πi::SimpleGamePolicy)(ai) = get(πi.p, ai, 0.0)

function (πi::SimpleGamePolicy)()
    D = SetCategorical(collect(keys(πi.p)), collect(values(πi.p)))
    return rand(D)
end

joint(X) = vec(collect(product(X...)))

joint(π, πi, i) = [i == j ? πi : πj for (j, πj) in enumerate(π)]

function utility(𝒫::SimpleGame, π, i)
    𝒜, R = 𝒫.𝒜, 𝒫.R
    p(a) = prod(πj(aj) for (πj, aj) in zip(π, a))
    return sum(R(a)[i]*p(a) for a in joint(𝒜))
end
```

Algorithm 24.2. A policy associated with an agent is represented by a dictionary that maps actions to probabilities. There are different ways to construct a policy. One way is to pass in a dictionary directory, in which case the probabilities are normalized. Another way is to pass in a generator that creates this dictionary. We can also construct a policy by passing in an action, in which case it assigns probability 1 to that action. If we have an individual policy `πi`, we can call `πi(ai)` to compute the probability the policy associates with action `ai`. If we call `πi()`, then it will return a random action according to that policy. We can use `joint(𝒜)` to construct the joint action space from `𝒜`. We can use `utility(𝒫, π, i)` to compute the utility associated with executing joint policy `π` in the game `𝒫` from the perspective of agent `i`.

```
function best_response(𝒫::SimpleGame, π, i)
    U(ai) = utility(𝒫, joint(π, SimpleGamePolicy(ai), i), i)
    ai = _argmax(U, 𝒫.𝒜[i])
    return SimpleGamePolicy(ai)
end
```

Algorithm 24.3. For a simple game `𝒫`, we can compute a deterministic best response for agent `i` given that the other agents are playing policies in `π`.

24.2.2 Softmax Response

We can use a *softmax response* to model how agent i will select their action.[4] As discussed in section 6.7, humans are often not perfectly rational optimizers of expected utility. The principle underlying the softmax response model is that (typically human) agents are more likely to make errors in their optimization when those errors are less costly. Given a *precision parameter* $\lambda \geq 0$, this model selects action a^i according to

$$\pi^i(a^i) \propto \exp(\lambda U^i(a^i, \boldsymbol{\pi}^{-i})) \tag{24.4}$$

As $\lambda \to 0$, the agent is insensitive to differences in utility, and they select actions uniformly at random. As $\lambda \to \infty$, the policy converges to a deterministic best response. We can treat λ as a parameter that can be learned from data using, for example, maximum likelihood estimation (section 4.1). This learning-based approach aims to be predictive of behavior rather than prescriptive of behavior, though having a predictive model of other human agents can be useful in building a system that prescribes optimal behavior. Algorithm 24.4 provides an implementation of softmax response.

[4] This kind of model is sometimes referred to as a *logit response* or *quantal response*. We introduced similar softmax models earlier in the context of directed exploration strategies for reinforcement learning (section 15.4).

```
function softmax_response(𝒫::SimpleGame, π, i, λ)
    𝒜i = 𝒫.𝒜[i]
    U(ai) = utility(𝒫, joint(π, SimpleGamePolicy(ai), i), i)
    return SimpleGamePolicy(ai ⇒ exp(λ*U(ai)) for ai in 𝒜i)
end
```

Algorithm 24.4. For a simple game `𝒫` and a particular agent `i`, we can compute the softmax response policy `πi` given the other agents are playing policies in `π`. This computation requires specifying the precision parameter `λ`.

24.3 Nash Equilibrium

Computing a best response for an agent requires that we hold the policies of all other agents constant. When we do not constrain the policies of the other agents, the meaning of an optimal joint policy becomes less clear. In the context of games, there are several notions for optimality or solution concepts, but the *Nash equilibrium* is perhaps the most widely known.[5] A Nash equilibrium is a joint policy $\boldsymbol{\pi}$ in which all agents are following a best response. In other words, a Nash equilibrium is a joint policy in which no agent has an incentive to unilaterally switch their policy.

[5] Named for the American mathematician John Forbes Nash, Jr. (1928–2015) who formalized the concept. J. Nash, "Non-Cooperative Games," *Annals of Mathematics*, pp. 286–295, 1951.

All games with a finite action space have at least one Nash equilibrium.[6] Multiple Nash equilibria can exist in a single game (exercise 24.2). Sometimes Nash equilibria may involve deterministic policies, but this is not always the case (see example 24.3). Computing a Nash equilibrium is *PPAD-complete*, a class that is distinct from NP-complete (appendix C.2) but also has no known polynomial time algorithm.[7]

[6] Exercise 24.1 explores the case where the action space is infinite.

[7] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, "The Complexity of Computing a Nash Equilibrium," *Communications of the ACM*, vol. 52, no. 2, pp. 89–97, 2009.

Example 24.3. Examples of deterministic and stochastic Nash equilibria.

Suppose we wish to find a Nash equilibrium for the prisoner's dilemma from example 24.1. If both agents always defect, both receive -3 reward. Any deviation by any agent will result in a -4 reward for that agent; hence, there is no incentive to deviate. Having both agents defect is thus a Nash equilibrium for the prisoner's dilemma.

Suppose we now wish to find a Nash equilibrium for rock-paper-scissors from example 24.2. Any deterministic strategy by one agent can be easily countered by the other agent. For example, if agent 1 plays rock then agent 2's best response is paper. Because there is no deterministic Nash equilibrium for rock-paper-scissors, we know there must be one involving stochastic policies. Suppose each agent selects from the actions uniformly at random. This solution produces an expected utility of 0 for both agents:

$$\begin{aligned} U^i(\boldsymbol{\pi}) = {} & 0\frac{1}{3}\frac{1}{3} - 1\frac{1}{3}\frac{1}{3} + 1\frac{1}{3}\frac{1}{3} \\ & + 1\frac{1}{3}\frac{1}{3} + 0\frac{1}{3}\frac{1}{3} - 1\frac{1}{3}\frac{1}{3} \\ & - 1\frac{1}{3}\frac{1}{3} + 1\frac{1}{3}\frac{1}{3} + 0\frac{1}{3}\frac{1}{3} \\ = {} & 0 \end{aligned}$$

Any deviation by an agent would decrease their expected payoff, meaning that we have found a Nash equilibrium.

The problem of finding a Nash equilibrium can be framed as an optimization problem:

$$\begin{aligned} \underset{\boldsymbol{\pi}, U}{\text{minimize}} \quad & \sum_i \left(U^i - U^i(\boldsymbol{\pi}) \right) \\ \text{subject to} \quad & U^i \geq U^i(a^i, \boldsymbol{\pi}^{-i}) \text{ for all } i, a^i \\ & \sum_{a^i} \pi^i(a^i) = 1 \text{ for all } i \\ & \pi^i(a^i) \geq 0 \text{ for all } i, a^i \end{aligned} \tag{24.5}$$

The optimization variables correspond to the parameters of $\boldsymbol{\pi}$ and U. At convergence, the objective will be 0 with U^i matching the utilities associated with policy $\boldsymbol{\pi}$ as computed in equation (24.1) for each agent i. The first constraint ensures that no agent will do better by unilaterally changing their action. Like the objective, this first constraint in nonlinear because it involves a product of the parameters in the optimization variable $\boldsymbol{\pi}$. The last two constraints are linear, ensuring that $\boldsymbol{\pi}$ represents a valid set of probability distributions over actions.

```
struct NashEquilibrium end

function tensorform(𝒫::SimpleGame)
    ℐ, 𝒜, R = 𝒫.ℐ, 𝒫.𝒜, 𝒫.R
    ℐ′ = eachindex(ℐ)
    𝒜′ = [eachindex(𝒜[i]) for i in ℐ]
    R′ = [R(a) for a in joint(𝒜)]
    return ℐ′, 𝒜′, R′
end

function solve(M::NashEquilibrium, 𝒫::SimpleGame)
    ℐ, 𝒜, R = tensorform(𝒫)
    model = Model(Ipopt.Optimizer)
    @variable(model, U[ℐ])
    @variable(model, π[i=ℐ, 𝒜[i]] ≥ 0)
    @NLobjective(model, Min,
        sum(U[i] - sum(prod(π[j,a[j]] for j in ℐ) * R[y][i]
            for (y,a) in enumerate(joint(𝒜))) for i in ℐ))
    @NLconstraint(model, [i=ℐ, ai=𝒜[i]],
        U[i] ≥ sum(
            prod(j==i ? (a[j]==ai ? 1.0 : 0.0) : π[j,a[j]] for j in ℐ)
            * R[y][i] for (y,a) in enumerate(joint(𝒜))))
    @constraint(model, [i=ℐ], sum(π[i,ai] for ai in 𝒜[i]) == 1)
    optimize!(model)
    πi′(i) = SimpleGamePolicy(𝒫.𝒜[i][ai] ⇒ value(π[i,ai]) for ai in 𝒜[i])
    return [πi′(i) for i in ℐ]
end
```

Algorithm 24.5. This nonlinear program computes a Nash equilibrium for a simple game $\mathcal{P}$.

24.4 Correlated Equilibrium

The *correlated equilibrium* generalizes the Nash equilibrium concept by relaxing the assumption that the agents act independently. The joint action in this case comes from a full joint distribution. A *correlated joint policy* $\pi(\mathbf{a})$ is a single distribution over the joint actions of all agents. Consequently, the actions of the various agents may be correlated, preventing the policies from being decoupled into individual policies $\pi^i(a^i)$. Algorithm 24.6 shows how to represent such a policy.

```
mutable struct JointCorrelatedPolicy
    p # dictionary mapping from joint actions to probabilities
    JointCorrelatedPolicy(p::Base.Generator) = new(Dict(p))
end

(π::JointCorrelatedPolicy)(a) = get(π.p, a, 0.0)

function (π::JointCorrelatedPolicy)()
    D = SetCategorical(collect(keys(π.p)), collect(values(π.p)))
    return rand(D)
end
```

Algorithm 24.6. A joint correlated policy is represented by a dictionary mapping joint actions to probabilities. If `π` is a joint correlated policy, evaluating `π(a)` will return the probability associated with the joint action `a`.

A *correlated equilibrium* is a correlated joint policy where no agent i can increase their expected utility by deviating from their current action a^i to another action $a^{i\prime}$:

$$\sum_{\mathbf{a}^{-i}} R^i(a^i, \mathbf{a}^{-i})\pi(a^i, \mathbf{a}^{-i}) \geq \sum_{\mathbf{a}^{-i}} R^i(a^{i\prime}, \mathbf{a}^{-i})\pi(a^i, \mathbf{a}^{-i}) \tag{24.6}$$

Example 24.4 demonstrates this concept.

Every Nash equilibrium is a correlated equilibrium because we can always form a joint policy from independent policies:

$$\pi(\mathbf{a}) = \prod_{i=1}^{k} \pi^i(a^i) \tag{24.7}$$

If the individual policies satisfy equation (24.2), then the joint policy will satisfy equation (24.6). Not all correlated equilibria, however, are Nash equilibria.

A correlated equilibrium can be computed using linear programming (algorithm 24.7):

Example 24.4. An example of computing correlated equilibria in rock-paper-scissors.

Consider again rock-paper-scissors from example 24.2. In example 24.3, we found that a Nash equilibrium is for both agents select their actions uniformly at random. In correlated equilibria, we use a correlated joint policy $\pi(\mathbf{a})$, meaning we need to find a distribution over (rock, rock), (rock, paper), (rock, scissors), (paper, rock), and so on. There are nine possible joint actions.

First, consider the joint policy in which agent 1 selects rock and agent 2 selects scissors. The utilities are

$$U^1(\pi) = 0\frac{0}{9} - 1\frac{0}{9} + 1\frac{9}{9} + 1\frac{0}{9} + \cdots = 1$$
$$U^2(\pi) = 0\frac{0}{9} + 1\frac{0}{9} - 1\frac{9}{9} - 1\frac{0}{9} + \cdots = -1$$

If agent 2 switched its action to paper, they would receive a utility of 1. Hence, this is not a correlated equilibrium.

Consider instead a correlated joint policy in which the joint action was chosen uniformly at random, with $\pi(\mathbf{a}) = 1/9$.

$$U^1(\pi) = 0\frac{1}{9} - 1\frac{1}{9} + 1\frac{1}{9} + 1\frac{1}{9} + \cdots = 0$$
$$U^2(\pi) = 0\frac{1}{9} + 1\frac{1}{9} - 1\frac{1}{9} - 1\frac{1}{9} + \cdots = 0$$

Any deviation from this results in one agent gaining utility and the other losing utility. This is a correlated equilibrium for rock-paper-scissors.

$$\begin{aligned}
\underset{\pi}{\text{maximize}} \quad & \sum_i \sum_{\mathbf{a}} R^i(\mathbf{a})\pi(\mathbf{a}) \\
\text{subject to} \quad & \sum_{\mathbf{a}^{-i}} R^i(a^i, \mathbf{a}^{-i})\pi(a^i, \mathbf{a}^{-i}) \geq \sum_{\mathbf{a}^{-i}} R^i(a^{i\prime}, \mathbf{a}^{-i})\pi(a^i, \mathbf{a}^{-i}) \quad \text{for all } i, a^i, a^{i\prime} \\
& \sum_{\mathbf{a}} \pi(\mathbf{a}) = 1 \\
& \pi(\mathbf{a}) \geq 0 \quad \text{for all } \mathbf{a}
\end{aligned} \tag{24.8}$$

Although linear programs can be solved in polynomial time, the size of the joint action space grows exponentially with the number of agents. The constraints enforce a correlated equilibrium. The objective, however, can be used to select between different valid correlated equilibria. Table 24.1 provides several common choices for the objective function. Not all of these are linear.

```
struct CorrelatedEquilibrium end

function solve(M::CorrelatedEquilibrium, 𝒫::SimpleGame)
    ℐ, 𝒜, R = 𝒫.ℐ, 𝒫.𝒜, 𝒫.R
    model = Model(Ipopt.Optimizer)
    @variable(model, π[joint(𝒜)] ≥ 0)
    @objective(model, Max, sum(sum(π[a]*R(a) for a in joint(𝒜))))
    @constraint(model, [i=ℐ, ai=𝒜[i], ai′=𝒜[i]],
        sum(R(a)[i]*π[a] for a in joint(𝒜) if a[i]==ai)
        ≥ sum(R(joint(a,ai′,i))[i]*π[a] for a in joint(𝒜) if a[i]==ai))
    @constraint(model, sum(π) == 1)
    optimize!(model)
    return JointCorrelatedPolicy(a ⇒ value(π[a]) for a in joint(𝒜))
end
```

Algorithm 24.7. Correlated equilibria are a more general notion of optimality for a simple game $\mathcal{P}$ than a Nash equilibrium. They can be computed using a linear program. The resulting policies are correlated, meaning that the agents stochastically select their joint actions.

Name	Description	Objective Function
Utilitarian	Maximize the net utility.	$\text{maximize}_\pi \sum_i \sum_{\mathbf{a}} R^i(\mathbf{a})\pi(\mathbf{a})$
Egalitarian	Maximize the minimum of all agents' utilities.	$\text{maximize}_\pi \, \text{minimize}_i \sum_{\mathbf{a}} R^i(\mathbf{a})\pi(\mathbf{a})$
Plutocratic	Maximize the maximum of all agents' utilities.	$\text{maximize}_\pi \, \text{maximize}_i \sum_{\mathbf{a}} R^i(\mathbf{a})\pi(\mathbf{a})$
Dictatorial	Maximize agent i's utility.	$\text{maximize}_\pi \sum_{\mathbf{a}} R^i(\mathbf{a})\pi(\mathbf{a})$

Table 24.1. Alternative objective functions for equation (24.8), which select for different correlated equilibria. These descriptions were adapted from A. Greenwald and K. Hall, "Correlated Q-Learning," in *International Conference on Machine Learning (ICML)*, 2003.

24.5 Iterated Best Response

Because computing a Nash equilibrium can be computationally expensive, an alternative approach is to iteratively apply best responses in a series of repeated games. In *iterated best response* (algorithm 24.8), we randomly cycle between agents, solving for each agent's best response policy in turn. This process may converge to a Nash equilibrium, but there are guarantees only for certain classes of games.[8] In many problems, it is common to observe cycles.

[8] Iterated best response will converge, for example, for a class known as *potential games*, as discussed in Theorem 19.12 of the textbook by N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, eds., *Algorithmic Game Theory*. Cambridge University Press, 2007.

```
struct IteratedBestResponse
    k_max # number of iterations
    π     # initial policy
end

function IteratedBestResponse(𝒫::SimpleGame, k_max)
    π = [SimpleGamePolicy(ai => 1.0 for ai in 𝒜i) for 𝒜i in 𝒫.𝒜]
    return IteratedBestResponse(k_max, π)
end

function solve(M::IteratedBestResponse, 𝒫)
    π = M.π
    for k in 1:M.k_max
        π = [best_response(𝒫, π, i) for i in 𝒫.ℐ]
    end
    return π
end
```

Algorithm 24.8. Iterated best response involves cycling through the agents and applying their best response to the other agents. The algorithm starts with some initial policy and stops after `k_max` iterations. For convenience, we have a constructor that takes as input a simple game and creates an initial policy that has each agent select actions uniformly at random. The same solve function will be reused in the next chapter in the context of more complicated forms of games.

24.6 Hierarchical Softmax

An area known as *behavioral game theory* aims to model human agents. When building decision-making systems that must interact with humans, computing the Nash equilibrium is not always helpful. Humans often do not play a Nash equilibrium strategy. First of all, it may be unclear which equilibrium to adopt if there are many different equilibria in the game. For games with only one equilibrium, it may be difficult for a human to compute the Nash equilibrium because of cognitive limitations. Even if human agents can compute the Nash equilibrium, they may doubt that their opponents can perform that computation.

There are many different behavioral models in the literature,[9] but one approach is to combine the iterated approach from the previous section with a softmax model. This *hierarchical softmax* approach (algorithm 24.9)[10] models the *depth of rationality* of an agent by a level $k \geq 0$. A level-0 agent plays its actions uniformly at random. A level-1 agent assumes the other players adopt level-0 strategies and selects actions according to a softmax response with precision λ. A level-k agent selects actions according to a softmax model of the other players playing level-$(k-1)$. Figure 24.1 illustrates this approach on a simple game.

[9] C. F. Camerer, *Behavioral Game Theory: Experiments in Strategic Interaction*. Princeton University Press, 2003.

[10] This approach is sometimes called *quantal-level-k* or *logit-level-k*. D. O. Stahl and P. W. Wilson, "Experimental Evidence on Players' Models of Other Players," *Journal of Economic Behavior & Organization*, vol. 25, no. 3, pp. 309–327, 1994.

```
struct HierarchicalSoftmax
    λ # precision parameter
    k # level
    π # initial policy
end

function HierarchicalSoftmax(𝒫::SimpleGame, λ, k)
    π = [SimpleGamePolicy(ai => 1.0 for ai in 𝒜i) for 𝒜i in 𝒫.𝒜]
    return HierarchicalSoftmax(λ, k, π)
end

function solve(M::HierarchicalSoftmax, 𝒫)
    π = M.π
    for k in 1:M.k
        π = [softmax_response(𝒫, π, i, M.λ) for i in 𝒫.ℐ]
    end
    return π
end
```

Algorithm 24.9. The hierarchical softmax model with precision parameter `λ` and level `k`. By default, it starts with an initial joint policy that assigns uniform probability to all individual actions.

We can learn the k and λ parameters of this behavioral model from data. If we have a collection of joint actions played by different agents, we can compute the associated likelihood for a given k and λ. We can then use an optimization algorithm to attempt to find a k and λ that maximizes likelihood. This optimization typically cannot be done analytically, but we can use numerical methods to perform this optimization.[11] Alternatively, we can use a Bayesian approach to parameter learning.[12]

[11] J. R. Wright and K. Leyton-Brown, "Beyond Equilibrium: Predicting Human Behavior in Normal Form Games," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

[12] J. R. Wright and K. Leyton-Brown, "Behavioral Game Theoretic Models: A Bayesian Framework for Parameter Analysis," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2012.

24.7 Fictitious Play

An alternative approach for computing policies for different agents is to have them play each other in simulation and learn how to best respond. Algorithm 24.10 provides an implementation of the simulation loop. At each iteration, we evaluate

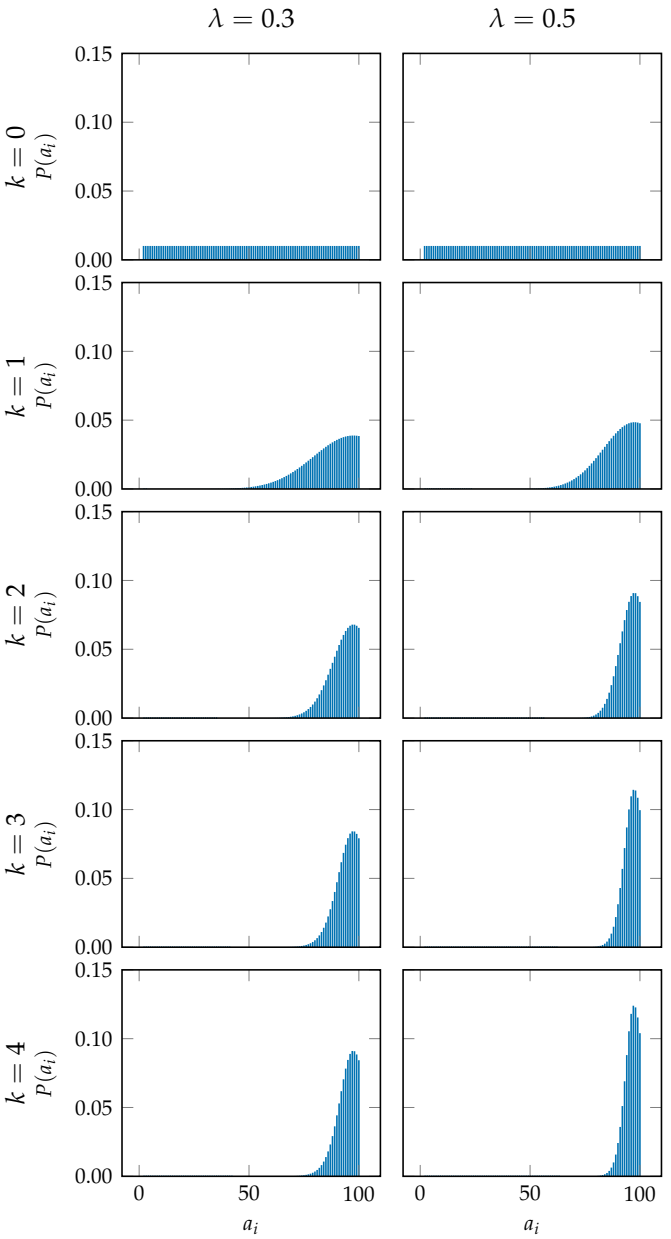

Figure 24.1. The hierarchical softmax model applied to the traveler's dilemma (appendix F.12) for different depths of rationality k and precision parameters λ. People tend to select actions between \$97 and \$100, even though the Nash equilibrium is only \$2.

the various policies to obtain a joint action, and then this joint action is used to by the agents to update their policies. We can use different ways to update the policies in response to observed joint actions. This section focuses on *fictitious play*, where the agents use maximum likelihood estimates (section 16.1) of the policies followed by the other agents. Each agent follows its own best response assuming the other agents act according to those estimates.[13]

[13] The term fictitious comes from how each agent plays against a fictitious strategy. G. W. Brown, "Iterative Solution of Games by Fictitious Play," *Activity Analysis of Production and Allocation*, vol. 13, no. 1, pp. 374–376, 1951. J. Robinson, "An Iterative Method of Solving a Game," *Annals of Mathematics*, pp. 296–301, 1951.

```
function simulate(𝒫::SimpleGame, π, k_max)
    for k = 1:k_max
        a = [πi() for πi in π]
        for πi in π
            update!(πi, a)
        end
    end
    return π
end
```

Algorithm 24.10. The simulation of a joint policy in simple game $\mathcal{P}$ for `k_max` iterations. The joint policy `π` is a vector of policies that can be individually updated through calls to `update!(πi, a)`.

To compute a maximum likelihood estimate, agent i tracks of the number of times agent j takes action a^j, storing it in a table $N^i(j, a^j)$. These counts can be initialized to any value, but they are often initialized to 1 to create initial uniform uncertainty. Agent i computes their best response assuming that each agent j follows the stochastic policy:

$$\pi^j(a^j) \propto N^i(j, a^j) \tag{24.9}$$

At each iteration, we have each agent act according to a best response assuming these stochastic count-based policies for the other agents. We then update the action counts for the actions taken. Algorithm 24.11 implements this simple adaptive procedure. Figure 24.2 and figure 24.3 show how agents' policies evolve over time using fictitious play. Fictitious play is not guaranteed to converge to a Nash equilibrium.[14]

There are many variants of fictitious play. One variant, called *smooth fictitious play*,[15] selects a best response using expected utility plus a smoothing function such as the entropy of the policy. Another variant is called *rational learning* or Bayesian learning. Rational learning expands the model of fictitious play to be any belief over other agent's actions, formulated as a Bayesian prior. Bayes' rule is then used to update the beliefs given the history of joint actions. Traditional fictitious play can be seen as rational learning with a Dirichlet prior (section 4.2.2).

[14] A concise background is provided by U. Berger, "Brown's Original Fictitious Play," *Journal of Economic Theory*, vol. 135, no. 1, pp. 572–578, 2007.

[15] D. Fudenberg and D. Levine, "Consistency and Cautious Fictitious Play," *Journal of Economic Dynamics and Control*, 1995.

```
mutable struct FictitiousPlay
    𝒫  # simple game
    i  # agent index
    N  # array of action count dictionaries
    πi # current policy
end

function FictitiousPlay(𝒫::SimpleGame, i)
    N = [Dict(aj => 1 for aj in 𝒫.𝒜[j]) for j in 𝒫.ℐ]
    πi = SimpleGamePolicy(ai ⇒ 1.0 for ai in 𝒫.𝒜[i])
    return FictitiousPlay(𝒫, i, N, πi)
end

(πi::FictitiousPlay)() = πi.πi()

(πi::FictitiousPlay)(ai) = πi.πi(ai)

function update!(πi::FictitiousPlay, a)
    N, 𝒫, ℐ, i = πi.N, πi.𝒫, πi.𝒫.ℐ, πi.i
    for (j, aj) in enumerate(a)
        N[j][aj] += 1
    end
    p(j) = SimpleGamePolicy(aj ⇒ u/sum(values(N[j])) for (aj, u) in N[j])
    π = [p(j) for j in ℐ]
    πi.πi = best_response(𝒫, π, i)
end
```

Algorithm 24.11. Fictitious play is a simple learning algorithm for an agent `i` of a simple game $\mathcal{P}$ that maintains `counts` of other agent action selections over time 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.

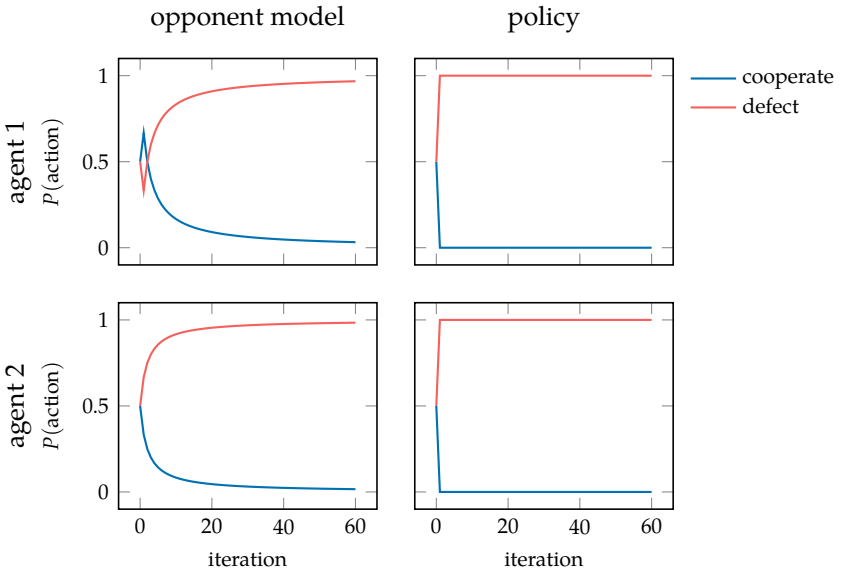

Figure 24.2. Two fictitious play agents learning and adapting to one another in a prisoner's dilemma game. The first row illustrates agent 1's learned model of 2 (left) and agent 1's policy (right) over iteration. The second row follows the same pattern but for agent 2. To illustrate variation in learning behavior, the initial counts for each agent's model over the other agent's action were assigned to a random number between 1 and 10.

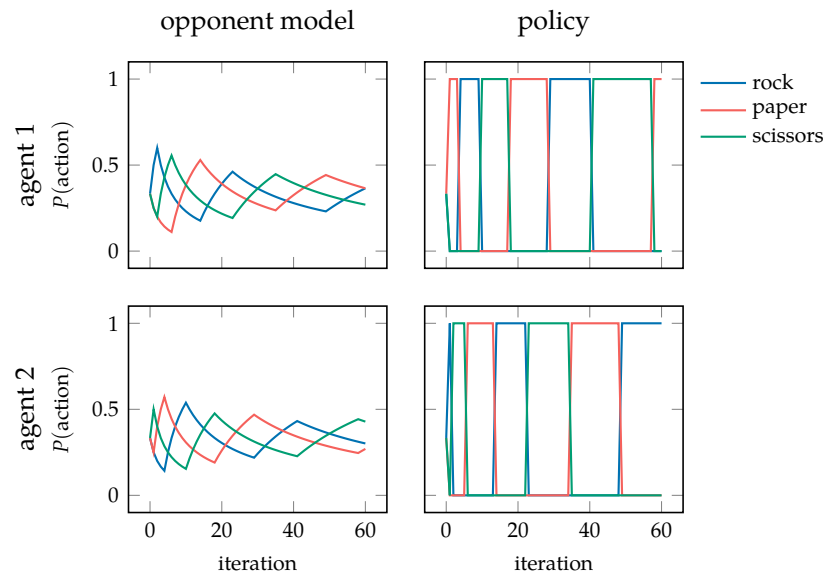

Figure 24.3. A visualization of two fictitious play agents learning and adapting to one another in a rock-paper-scissors game. The first row illustrates agent 1's learned model of 2 (left) and agent 1's policy (right) over time. The second row follows the same pattern but for agent 2. To illustrate variation in learning behavior, the initial counts for each agent's model over the other agent's action were assigned to a random number between 1 and 10. In this zero-sum game, fictitious play agents approach convergence to their stochastic policy Nash equilibrium.

24.8 Summary

- In simple games, multiple agents compete to maximize expected reward.
- Optimality is not as straightforward in the multiagent setting, with multiple possible solution concepts for extracting policies from a reward specification.
- A best response of an agent to a fixed set of policies of the other agents is one where there is no incentive to deviate.
- A Nash equilibrium is a joint policy where all agents follow a best response.
- A correlated equilibrium is the same as a Nash equilibrium, except all agents follow a single joint action distribution that allows for correlation between agents.
- Iterated best response can quickly optimize a joint policy by iteratively applying best responses, but there are no general guarantees of convergence.
- Hierarchical softmax attempts to model agents in terms of their depth of rationality and precision, which can be learned from past joint actions.
- Fictitious play is a learning algorithm that uses maximum-likelihood action models for other agents to find best response policies, with the potential to converge to a Nash equilibrium.

24.9 Exercises

Exercise 24.1. Give an example of a game with two agents and an infinite number of actions such that a Nash equilibrium does not exist.

Solution: Suppose the action space of each agent consists of the negative real numbers and their reward is equal to their action. Since no greatest negative number exists, a Nash equilibrium cannot exist.

Exercise 24.2. Give an example of a game with two agents, two actions, and two Nash equilibria involving deterministic policies.

Solution: Here is one example.[16] Suppose we have two aircraft on a collision course, and the pilots of each aircraft must choose between climb or descend to avoid collision. If the pilots both choose the same maneuver, then there is a crash with utility -4 to both pilots. Because climbing requires more fuel than descending, there is an additional penalty of -1 to any pilot who decides to climb.

[16] This example comes from M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

agent 1 \ agent 2	descend	climb
climb	$-5, -5$	$-1, 0$
descend	$0, -1$	$-4, -4$

Exercise 24.3. Given a stationary joint policy $\boldsymbol{\pi}$ that is a Nash equilibrium for a simple game with horizon of 1, prove that it is also a Nash equilibrium for the same simple game repeated to any finite or infinite horizon.

Solution: By definition of the Nash equilibrium in equation (24.2), all agents i are performing a best response π^i:

$$U^i(\pi^i, \boldsymbol{\pi}^{-i}) \geq U^i(\pi^{i\prime}, \boldsymbol{\pi}^{-i})$$

to all other policies $\pi^{i\prime} \neq \pi^i$. By definition of U^i, we have

$$U^i(\boldsymbol{\pi}) = \sum_{\mathbf{a} \in \mathbf{A}} R^i(\mathbf{a}) \prod_{j=1}^{k} \pi^j(a^j)$$

The joint policy remains constant over time for all agents. Apply any horizon n, with any discount factor ($\gamma = 1$ for $n < \infty$; $\gamma < 1$ for $n \to \infty$). The utility of agent i after n steps is

$$\begin{aligned} U^{i,n}(\boldsymbol{\pi}) &= \sum_{t=1}^{n} \gamma^{t-1} \sum_{\mathbf{a} \in \mathbf{A}} R^i(\mathbf{a}) \prod_{j=1}^{k} \pi^j(a^j) \\ &= \sum_{\mathbf{a} \in \mathbf{A}} R^i(\mathbf{a}) \prod_{j=1}^{k} \pi^j(a^j) \sum_{t=1}^{n} \gamma^{t-1} \\ &= U^i(\boldsymbol{\pi}) \sum_{t=1}^{n} \gamma^{t-1} \\ &= U^i(\boldsymbol{\pi}) c \end{aligned}$$

The discount factor becomes a constant multiplier $c > 0$. Therefore, any constant multiplication of equation (24.2) on both sides results in the same inequality, completing the

proof:

$$\begin{aligned}
U^i(\pi^i, \boldsymbol{\pi}^{-i}) &\geq U^i(\pi^{i\prime}, \boldsymbol{\pi}^{-i}) \\
U^i(\pi^i, \boldsymbol{\pi}^{-i})c &\geq U^i(\pi^{i\prime}, \boldsymbol{\pi}^{-i})c \\
U^i(\pi^i, \boldsymbol{\pi}^{-i}) \sum_{t=1}^{n} \gamma^{t-1} &\geq U^i(\pi^{i\prime}, \boldsymbol{\pi}^{-i}) \sum_{t=1}^{n} \gamma^{t-1} \\
\sum_{t=1}^{n} \gamma^{t-1} U^i(\pi^i, \boldsymbol{\pi}^{-i}) &\geq \sum_{t=1}^{n} \gamma^{t-1} U^i(\pi^{i\prime}, \boldsymbol{\pi}^{-i}) \\
U^{i,n}(\pi^i, \boldsymbol{\pi}^{-i}) &\geq U^{i,n}(\pi^{i\prime}, \boldsymbol{\pi}^{-i})
\end{aligned}$$

Exercise 24.4. Prove that a Nash equilibrium is a correlated equilibrium.

Solution: Given any Nash equilibrium $\boldsymbol{\pi}$, define a correlated joint policy $\pi(\mathbf{a})$ such that

$$\pi(\mathbf{a}) = \prod_{i=1}^{k} \pi^i(a^i)$$

By definition of a Nash equilibrium, for any agent i and any other policy $\pi^{i\prime} \neq \pi^i$, we have a best response. Applying the definition of best response from equation (24.2), utility from equation (24.1), and this correlated joint policy $\pi(\mathbf{a})$, we obtain:

$$\begin{aligned}
U^i(\pi^i, \boldsymbol{\pi}^{-i}) &\geq U^i(\pi^{i\prime}, \boldsymbol{\pi}^{-i}) \\
\sum_{\mathbf{a}} R^i(\mathbf{a}) \pi^i(a^i) \prod_{j \neq i} \pi^j(a^j) &\geq \sum_{\mathbf{a}} R^i(\mathbf{a}) \pi^{i\prime}(a^i) \prod_{j \neq i} \pi^j(a^j) \\
\sum_{a^i} \pi^i(a^i) \sum_{\mathbf{a}} R^i(\mathbf{a}) \prod_{j \neq i} \pi^j(a^j) &\geq \sum_{a^i} \pi^{i\prime}(a^i) \sum_{\mathbf{a}} R^i(\mathbf{a}) \prod_{j \neq i} \pi^j(a^j)
\end{aligned}$$

Consider each element in the summation over a^i, and consider any action with positive likelihood $\pi^i(a^i) > 0$ and divide each side by it. The left side is 1. The right side is either < 1 or $\pi^i(a^i) < \pi^{i\prime}(a^i)$, in which case the inequality can be preserved by simply $\frac{\pi^i(a^i)}{\pi^{i\prime}(a^i)} \geq 1$. The result is that the term falls out.

Consider the following equation, for any deterministically chosen action a^i such that $\pi^i(a^i) = 1$ and any other action $a^{i\prime}$:

$$\begin{aligned}
\sum_{\mathbf{a}^{-i}} R^i(a^i, \mathbf{a}^{-i}) \pi^{-i}(\mathbf{a}^{-i}) &\geq \sum_{\mathbf{a}^{-i}} R^i(a^{i\prime}, \mathbf{a}^{-i}) \pi^{-i}(\mathbf{a}^{-i}) \\
\pi^i(a^i) \sum_{\mathbf{a}^{-i}} R^i(a^i, \mathbf{a}^{-i}) \pi^{-i}(\mathbf{a}^{-i}) &\geq \pi^i(a^i) \sum_{\mathbf{a}^{-i}} R^i(a^{i\prime}, \mathbf{a}^{-i}) \pi^{-i}(\mathbf{a}^{-i}) \\
\sum_{\mathbf{a}^{-i}} R^i(a^i, \mathbf{a}^{-i}) \pi(\mathbf{a}) &\geq \sum_{\mathbf{a}^{-i}} R^i(a^{i\prime}, \mathbf{a}^{-i}) \pi(\mathbf{a})
\end{aligned}$$

Any mixing of these equations over actions trivially holds true. This satisfies equation (24.6).

Exercise 24.5. Give an example two-agent game, each with two actions, for which the correlated equilibria cannot be represented as a Nash equilibrium.

Solution: Consider the following game in which two people want to go on a date but have a conflicting preference on what kind of date, in this case a dinner or a movie.

		agent 2: dinner	agent 2: movie
agent 1	dinner	2, 1	0, 0
agent 1	movie	0, 0	1, 2

A Nash equilibrium here is to simply uniformly at random choose dinner or movie, $\pi^i(\text{dinner}) = \pi^i(\text{movie}) = 0.5$ for both agents i. This results in an expected utilities of 2/3:

$$U^1(\boldsymbol{\pi}) = 0.5 \cdot 0.5 \cdot 2 + 0.5 \cdot 0.5 \cdot 0 + 0.5 \cdot 0.5 \cdot 0 + 0.5 \cdot 0.5 \cdot 1 = 0.25 \cdot 2 + 0.25 \cdot 1 = \frac{2}{3}$$
$$U^2(\boldsymbol{\pi}) = 0.5 \cdot 0.5 \cdot 1 + 0.5 \cdot 0.5 \cdot 0 + 0.5 \cdot 0.5 \cdot 0 + 0.5 \cdot 0.5 \cdot 2 = 0.25 \cdot 1 + 0.25 \cdot 2 = \frac{2}{3}$$

This is of course a correlated equilibrium as well.

However, if the two agents correlated their actions on a fair coin flip $\pi(\text{movie}, \text{movie}) = \pi(\text{dinner}, \text{dinner}) = 0.5$, then they could coordinate either both going to the dinner or both going to the movies.

$$U^1(\boldsymbol{\pi}) = 0.5 \cdot 2 + 0.0 \cdot 0 + 0.0 \cdot 0 + 0.5 \cdot 1 = 0.5 \cdot 2 + 0.5 \cdot 1 = \frac{3}{2}$$
$$U^2(\boldsymbol{\pi}) = 0.5 \cdot 1 + 0.5 \cdot 0 + 0.5 \cdot 0 + 0.5 \cdot 2 = 0.5 \cdot 1 + 0.5 \cdot 2 = \frac{3}{2}$$

This is not possible with a Nash equilibrium. Intuitively, in this example, this is because the probabilistic weight is spread out over each row, independently for each player. Conversely, a correlated equilibrium can be targeted toward a specific cell, in this case able to obtain a higher payoff.

Exercise 24.6. Algorithms such as iterated best response and fictitious play do not converge in every game. Construct a game that demonstrates this non-convergence.

Solution: Iterated best response diverges in rock-paper-scissors. Fictitious play also will not converge in almost-rock-paper-scissors:[17]

[17] This game and many others are discussed in greater detail by Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

agent 1 \ agent 2	rock	paper	scissors
rock	0, 0	0, 1	1, 0
paper	1, 0	0, 0	0, 1
scissors	0, 1	1, 0	0, 0

The general pattern of convergence requires a shared interest. Consider the game in the solution to exercise 24.5. Both agents have a vested interest in going on a date together. This class of game is called a *potential game*.

Exercise 24.7. What does iterated best response converge to in the traveler's dilemma (appendix F.12)?

Solution: It converges to the Nash equilibrium of $2.