6

Richer Representations: Beyond the Normal and Extensive Forms

In this chapter we will go beyond the normal and extensive forms by considering a variety of richer game representations. These further representations are important because the normal and extensive forms are not always suitable for modeling large or realistic game-theoretic settings.

First, we may be interested in games that are not finite and that therefore cannot be represented in normal or extensive form. For example, we may want to consider what happens when a simple normal-form game such as the Prisoner's Dilemma is repeated infinitely. We might want to consider a game played by an uncountably infinite set of agents. Or we may want to use an interval of the real numbers as each player's action space.[1]

Second, both of the representations we have studied so far presume that agents have perfect knowledge of everyone's payoffs. This seems like a poor model of many realistic situations, where, for example, agents might have private information that affects their own payoffs and other agents might have only probabilistic information about each others' private information. An elaboration like this can have a big impact, because one agent's actions can depend on what he knows about another agent's payoffs.

Finally, as the numbers of players and actions in a game grow—even if they remain finite—games can quickly become far too large to reason about or even to write down using the representations we have studied so far. Luckily, we are not usually interested in studying arbitrary strategic situations. The sorts of noncooperative settings that are most interesting in practice tend to involve highly structured payoffs. This can occur because of constraints imposed by the fact that the play of a game actually unfolds over time (e.g., because a large game actually corresponds to finitely repeated play of a small game). It can also occur because of the nature of the problem domain (e.g., while the world may involve many agents, the number of agents who are able to directly affect any given agent's payoff is small). If we understand the way in which agents' payoffs are structured, we can represent them much more compactly than we would be able to do using

1. We will explore the first example in detail in this chapter. A thorough treatment of infinite sets of players or action spaces is beyond the scope of this book; nevertheless, we will consider certain games with infinite sets of players in Section 6.4.4 and with infinite action spaces in Chapters 10 and 11.

the normal or extensive forms. Often, these compact representations also allow us to reason more efficiently about the games they describe (e.g., the computation of Nash equilibria can be provably faster, or pure-strategy Nash equilibria can be proved to always exist).

In this chapter we will present various different representations that address these limitations of the normal and extensive forms. In Section 6.1 we will begin by considering the special case of extensive-form games that are constructed by repeatedly playing a normal-form game and then we will extend our consideration to the case where the normal form is repeated infinitely. This will lead us to stochastic games in Section 6.2, which are like repeated games but do not require that the same normal-form game is played in each time step. In Section 6.3 we will consider structure of a different kind: instead of considering time, we will consider games involving uncertainty. Specifically, in Bayesian games agents face uncertainty—and hold private information—about the game's payoffs. Section 6.4 describes congestion games, which model situations in which agents contend for scarce resources. Finally, in Section 6.5 we will consider representations that are motivated primarily by compactness and by their usefulness for permitting efficient computation (e.g., of Nash equilibria). Such compact representations can extend any other existing representation, such as normal-form games, extensive-form games, or Bayesian games.

6.1 Repeated games

In repeated games, a given game (often thought of in normal form) is played multiple times by the same set of players. The game being repeated is called the *stage game*. For example, Figure 6.1 depicts two players playing the Prisoner's Dilemma exactly twice in a row.

stage game

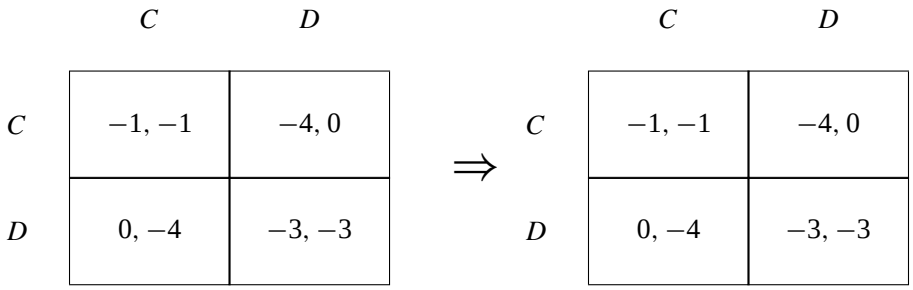

Figure 6.1 Twice-played Prisoner's Dilemma.

This representation of the repeated game, while intuitive, obscures some key factors. Do agents see what the other agents played earlier? Do they remember what they knew? And, while the utility of each stage game is specified, what is the utility of the entire repeated game?

We answer these questions in two steps. We first consider the case in which the game is repeated a finite and commonly-known number of times. Then we

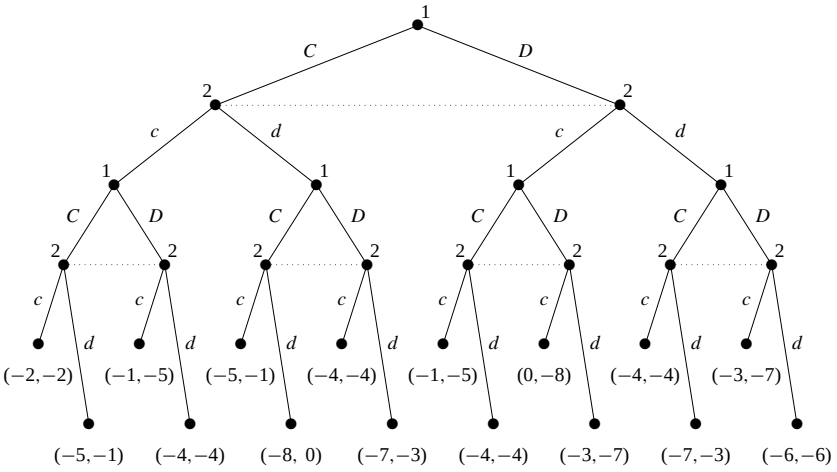

Figure 6.2 Twice-played Prisoner's Dilemma in extensive form.

consider the case in which the game is repeated infinitely often, or a finite but unknown number of times.

6.1.1 *Finitely repeated games*

One way to completely disambiguate the semantics of a finitely repeated game is to specify it as an imperfect-information game in extensive form. Figure 6.2 describes the twice-played Prisoner's Dilemma game in extensive form. Note that it captures the assumption that at each iteration the players do not know what the other player is playing, but afterward they do. Also note that the payoff function of each agent is additive; that is, it is the sum of payoffs in the two-stage games.

The extensive form also makes it clear that the strategy space of the repeated game is much richer than the strategy space in the stage game. Certainly one strategy in the repeated game is to adopt the same strategy in each stage game; clearly, this memory less strategy, called a *stationary strategy*, is a behavioral strategy in the extensive-form representation of the game. But in general, the action (or mixture of actions) played at a stage game can depend on the history of play thus far. Since this fact plays a particularly important role in infinitely repeated games, we postpone further discussion of it to the next section. Indeed, in the finite, known repetition case, we encounter again the phenomenon of backward induction, which we first encountered when we introduced subgame-perfect equilibria. Recall that in the Centipede game, discussed in Section 5.1.3, the unique SPE was to go down and terminate the game at every node. Now consider a finitely repeated Prisoner's Dilemma game. Again, it can be argued, in the last round it is a dominant strategy to defect, no matter what happened so far. This is common knowledge, and no choice of action in the preceding rounds will impact the play in the last round. Thus in the second-to-last round too it is a dominant strategy to defect. Similarly, by induction, it can be argued that the only equilibrium in this case is to always defect. However, as in the case of the

stationary strategy

Centipede game, this argument is vulnerable to both empirical and theoretical criticisms.

6.1.2 *Infinitely repeated games*

When the infinitely repeated game is transformed into extensive form, the result is an infinite tree. So the payoffs cannot be attached to any terminal nodes, nor can they be defined as the sum of the payoffs in the stage games (which in general will be infinite). There are two common ways of defining a player's payoff in an infinitely repeated game to get around this problem. The first is the average payoff of the stage game in the limit.[2]

average reward

Definition 6.1.1 (Average reward) *Given an infinite sequence of payoffs* $r_i^{(1)}, r_i^{(2)}, \ldots$ *for player* i*, the* average reward *of* i *is*

$$\lim_{k\to\infty} \frac{\sum_{j=1}^{k} r_i^{(j)}}{k}.$$

The *future discounted reward* to a player at a certain point of the game is the sum of his payoff in the immediate stage game, plus the sum of future rewards discounted by a constant factor. This is a recursive definition, since the future rewards again give a higher weight to early payoffs than to later ones.

future discounted reward

Definition 6.1.2 (Discounted reward) *Given an infinite sequence of payoffs* $r_i^{(1)}, r_i^{(2)}, \ldots$ *for player* i*, and a discount factor* β *with* $0 \leq \beta \leq 1$*, the* future discounted reward *of* i *is* $\sum_{j=1}^{\infty} \beta^j r_i^{(j)}$.

The discount factor can be interpreted in two ways. First, it can be taken to represent the fact that the agent cares more about his well-being in the near term than in the long term. Alternatively, it can be assumed that the agent cares about the future just as much as he cares about the present, but with some probability the game will be stopped any given round; $1 - \beta$ represents that probability. The analysis of the game is not affected by which perspective is adopted.

Now let us consider strategy spaces in an infinitely repeated game. In particular, consider the infinitely repeated Prisoner's Dilemma game. As we discussed, there are many strategies other than stationary ones. One of the most famous is *Tit-for-Tat*. TfT is the strategy in which the player starts by cooperating and thereafter chooses in round $j + 1$ the action chosen by the other player in round j. Beside being both simple and easy to compute, this strategy is notoriously hard to beat; it was the winner in several repeated Prisoner's Dilemma competitions for computer programs.

Tit-for-Tat (TfT)

Since the space of strategies is so large, a natural question is whether we can characterize all the Nash equilibria of the repeated game. For example, if the discount factor is large enough, both players playing TfT is a Nash equilibrium. But there is an infinite number of others. For example, consider the *trigger strategy*. This is a draconian version of TfT; in the trigger strategy, a player starts

trigger strategy

2. The observant reader will notice a potential difficulty in this definition, since the limit may not exist. One can extend the definition to cover these cases by using the lim sup operator in Definition 6.1.1 rather than lim.

by cooperating, but if ever the other player defects then the first defects forever. Again, for sufficiently large discount factor, the trigger strategy forms a Nash equilibrium not only with itself but also with TfT.

The folk theorem—so-called because it was part of the common lore before it was formally written down—helps us understand the space of all Nash equilibria of an infinitely repeated game, by answering a related question. It does not characterize the equilibrium strategy profiles, but rather the payoffs obtained in them. Roughly speaking, it states that in an infinitely repeated game the set of average rewards attainable in equilibrium are precisely those pairs attainable under mixed strategies in a single-stage game, with the constraint on the mixed strategies that each player's payoff is at least the amount he would receive if the other players adopted minmax strategies against him.

More formally, consider any n-player game $G = (N, A, u)$ and any payoff profile $r = (r_1, r_2, \ldots, r_n)$. Let

$$v_i = \min_{s_{-i} \in S_{-i}} \max_{s_i \in S_i} u_i(s_{-i}, s_i).$$

In words, v_i is player i's minmax value: his utility when the other players play minmax strategies against him, and he plays his best response.

Before giving the theorem, we provide some more definitions.

Definition 6.1.3 (Enforceable) *A payoff profile $r = (r_1, r_2, \ldots, r_n)$ is* enforceable *if $\forall i \in N, r_i \geq v_i$.*

Definition 6.1.4 (Feasible) *A payoff profile $r = (r_1, r_2, \ldots, r_n)$ is* feasible *if there exist rational, nonnegative values α_a such that for all i, we can express r_i as $\sum_{a \in A} \alpha_a u_i(a)$, with $\sum_{a \in A} \alpha_a = 1$.*

In other words, a payoff profile is feasible if it is a convex, rational combination of the outcomes in G.

folk theorem

Theorem 6.1.5 (Folk Theorem) *Consider any n-player normal-form game G and any payoff profile $r = (r_1, r_2, \ldots, r_n)$.*

1. *If r is the payoff profile for any Nash equilibrium s of the infinitely repeated G with average rewards, then for each player i, r_i is enforceable.*
2. *If r is both feasible and enforceable, then r is the payoff profile for some Nash equilibrium of the infinitely repeated G with average rewards.*

This proof is both instructive and intuitive. The first part uses the definition of minmax and best response to show that an agent can never receive less than his minmax value in any equilibrium. The second part shows how to construct an equilibrium that yields each agent the average payoffs given in any feasible and enforceable payoff profile r. This equilibrium has the agents cycle in perfect lock-step through a sequence of game outcomes that achieve the desired average payoffs. If any agent deviates, the others punish him forever by playing their minmax strategies against him.

Proof. Part 1: Suppose r is not enforceable, that is, $r_i < v_i$ for some i. Then consider an alternative strategy for i: playing $BR(s_{-i}(h))$, where $s_{-i}(h)$ is the equilibrium strategy of other players given the current history h and $BR(s_{-i}(h))$ is a function that returns a best response for i to a given strategy profile s_{-i} in the (unrepeated) stage game G. By definition of a minmax strategy, player i receives a payoff of at least v_i in every stage game if he plays $BR(s_{-i}(h))$, and so i's average reward is also at least v_i. Thus, if $r_i < v_i$ then s cannot be a Nash equilibrium.

Part 2: Since r is a feasible enforceable payoff profile, we can write it as $r_i = \sum_{a \in A}(\frac{\beta_a}{\gamma})u_i(a)$, where β_a and γ are nonnegative integers. (Recall that α_a were required to be rational. So we can take γ to be their common denominator.) Since the combination was convex, we have $\gamma = \sum_{a \in A} \beta_a$.

We are going to construct a strategy profile that will cycle through all outcomes $a \in A$ of G with cycles of length γ, each cycle repeating action a exactly β_a times. Let (a^t) be such a sequence of outcomes. Let us define a strategy s_i of player i to be a trigger version of playing (a^t): if nobody deviates, then s_i plays a_i^t in period t. However, if there was a period t' in which some player $j \neq i$ deviated, then s_i will play $(p_{-j})_i$, where (p_{-j}) is a solution to the minimization problem in the definition of v_j.

First observe that if everybody plays according to s_i, then, by construction, player i receives average payoff of r_i (look at averages over periods of length γ). Second, this strategy profile is a Nash equilibrium. Suppose everybody plays according to s_i, and player j deviates at some point. Then, forever after, player j will receive his min max payoff $v_j \leq r_j$, rendering the deviation unprofitable. ■

The reader might wonder why this proof appeals to i's minmax value rather than his maxmin value. First, notice that the trigger strategies in Part 2 of the proof use minmax strategies to punish agent i. This makes sense because even in cases where i's minmax value is strictly greater than his maxmin value,[3] i's minmax value is the smallest amount that the other agents can guarantee that i will receive. When i best responds to a minmax strategy played against him by $-i$, he receives exactly his minmax value; this is the deviation considered in Part 1.

Theorem 6.1.5 is actually an instance of a large family of folk theorems. As stated, Theorem 6.1.5 is restricted to infinitely repeated games, to average reward, to the Nash equilibrium, and to games of complete information. However, there are folk theorems that hold for other versions of each of these conditions, as well as other conditions not mentioned here. In particular, there are folk theorems for infinitely repeated games with discounted reward (for a large enough discount factor), for finitely repeated games, for subgame-perfect equilibria (i.e., where agents only administer finite punishments to deviators), and for games of incomplete information. We do not review them here, but the message of each of them

3. This can happen in games with more than two players, as discussed in Section 3.4.1.

	C	D
C	3, 3	0, 4
D	4, 0	1, 1

Figure 6.3 Prisoner's Dilemma game.

is fundamentally the same: the payoffs in the equilibria of a repeated game are essentially constrained only by enforceability and feasibility.

6.1.3 *"Bounded rationality": repeated games played by automata*

Until now we have assumed that players can engage in arbitrarily deep reasoning and mutual modeling, regardless of their complexity. In particular, consider the fact that we have tended to rely on equilibrium concepts as predictions of—or prescriptions for—behavior. Even in the relatively uncontroversial case of two-player zero-sum games, this is a questionable stance in practice; otherwise, for example, there would be no point in chess competitions. While we will continue to make this questionable assumption in much of the remainder of the book, we pause here to revisit it. We ask what happens when agents are not perfectly rational expected-utility maximizers. In particular, we ask what happens when we impose specific computational limitations on them.

Consider (yet again) an instance of the Prisoner's Dilemma, which is reproduced in Figure 6.3. In the finitely repeated version of this game, we know that each player's dominant strategy (and thus the only Nash equilibrium) is to choose the strategy D in each iteration of the game. In reality, when people actually play the game, we typically observe a significant amount of cooperation, especially in the earlier iterations of the game. While much of game theory is open to the criticism that it does not match well with human behavior, this is a particularly stark example of this divergence. What models might explain this fact?

One early proposal in the literature is based on the notion of an ϵ-equilibrium, defined in Section 3.4.7. Recall that this is a strategy profile in which no agent can gain more than ϵ by changing his strategy; a Nash equilibrium is thus the special case of a 0-equilibrium. This equilibrium concept is motivated by the idea that agents' rationality may be bounded in the sense that they are willing to settle for payoffs that are slightly below their best response payoffs. In the finitely repeated Prisoner's Dilemma game, as the number of repetitions increases, the corresponding sets of ϵ-equilibria include outcomes with longer and longer sequences of the "cooperate" strategy.

Various other models of bounded rationality exist, but we will focus on what has proved to be the richest source of results so far, namely, restricting agents' strategies to those implemented by automata of the sort investigated in computer science.

Finite-state automata

The motivation for using automata becomes apparent when we consider the representation of a strategy in a repeated game. Recall that a finitely repeated game is an imperfect-information extensive-form game, and that a strategy for player i in such a game is a specification of an action for every information set belonging to that player. A strategy for k repetitions of an m-action game is thus a specification of $\frac{m^k-1}{m-1}$ different actions. However, a naive encoding of a strategy as a table mapping each possible history to an action can be extremely inefficient. For example, the strategy of choosing D in every round can be represented using just the single-stage strategy D, and the *Tit-for-Tat* strategy can be represented simply by specifying that the player mimic what his opponent did in the previous round. One representation that exploits this structure is the *finite-state automaton*, or *Moore machine*. The formal definition of a finite-state automaton in the context of a repeated game is as follows.

finite-state automaton

Moore machine

Definition 6.1.6 (Automaton) *Given a game $G = (N, A, u)$ that will be played repeatedly, an automaton M_i for player i is a four-tuple $(Q_i, q_i^0, \delta_i, f_i)$, where:*

- *Q_i is a set of states;*
- *q_i^0 is the start state;*
- *$\delta_i : Q_i \times A \mapsto Q_i$ is a transition function mapping the current state and an action profile to a new state; and*
- *$f_i : Q_i \mapsto A_i$ is a strategy function associating with every state an action for player i.*

An automaton is used to represent each player's repeated game strategy as follows. The machine begins in the start state q_i^0, and in the first round plays the action given by $f_i(q_i^0)$. Using the transition function and the actions played by the other players in the first round, it then transitions automatically to the new state $\delta_i(q_i^0, a_1, \ldots, a_n)$ before the beginning of round 2. It then plays the action $f_i(\delta_i(q_i^0, a_1, \ldots, a_n))$ in round two, and so on. More generally, we can specify the current strategy and state at round t using the following recursive definitions.

$$a_i^t = f_i(q_i^t)$$
$$q_i^{t+1} = \delta_i(q_i^t, a_1^t, \ldots, a_n^t)$$

Automaton representations of strategies are very intuitive when viewed graphically. The following figures show compact automaton representations of some common strategies for the repeated Prisoner's Dilemma game. Each circle is a state in the automaton and its label is the action to play at that state. The transitions are represented as labeled arrows. From the current state, we transition along the arrow labeled with the move the opponent played in the current game. The unlabeled arrow enters the initial state.

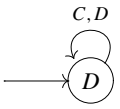

Figure 6.4 An automaton representing the repeated *Defect* action.

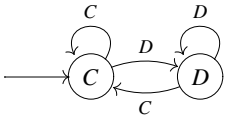

Figure 6.5 An automaton representing the *Tit-for-Tat* strategy.

The automaton represented by Figure 6.4 plays the constant D strategy, while Figure 6.5 encodes the more interesting *Tit-for-Tat* strategy. It starts in the C state, and the transitions are constructed so that the automaton always mimics the opponent's last action.

machine game

We can now define a new class of games, called *machine games*, in which each player selects an automaton representing a repeated game strategy.

Definition 6.1.7 (Machine game) *A two-player machine game* $G^M = (\{1, 2\}, \mathcal{M}, G)$ *of the* k*-period repeated game* G *is defined by:*

- *a pair of players* $\{1, 2\}$*;*
- $\mathcal{M} = \mathcal{M}_1, \mathcal{M}_2$*, where* $\mathcal{M}_1$ *is a set of available automata for player* i*; and*
- *a normal-form game* $G = (\{1, 2\}, A, u)$*.*

A pair $M_1 \in \mathcal{M}_1$ *and* $M_2 \in \mathcal{M}_2$ *deterministically yield an outcome* $o^t(M_1, M_2)$ *at each iteration* t *of the repeated game. Thus,* G^M *induces a normal-form game* $(\{1, 2\}, \mathcal{M}, U)$*, in which each player* i *chooses an automaton* $M_i \in \mathcal{M}_i$*, and obtains utility* $U_i(M_1, M_2) = \sum_{t=1}^{k} u_i(o^t(M_1, M_2))$*.*

Note that we can easily replace the k-period repeated game with a discounted (or limit of means) infinitely repeated game, with a corresponding change to $U_i(M_1, M_2)$ in the induced normal-form game.

In what follows, the function $s : M \mapsto \mathbb{Z}$ represents the number of states of an automaton M, and the function $S(\mathcal{M}_i) = \max_{M \in \mathcal{M}_i} s(M)$ represents the size of the largest automaton among a set of automata $\mathcal{M}_i$.

Automata of bounded size

Intuitively, automata with fewer states represent *simpler* strategies. Thus, one way to bound the rationality of the player is by limiting the number of states in the automaton.

Placing severe restrictions on the number of states not only induces an equilibrium in which cooperation always occurs, but also causes the always-defect equilibrium to disappear. This equilibrium in a finitely repeated Prisoner's Dilemma game depends on the assumption that each player can use *backward induction* (see Section 5.1.4) to find his dominant strategy. In order to perform backward induction in a k-period repeated game, each player needs to keep

track of at least k distinct states: one state to represent the choice of strategy in each repetition of the game. In the Prisoner's Dilemma, it turns out that if $2 < \max(S(\mathcal{M}_1), S(\mathcal{M}_2)) < k$, then the constant-defect strategy does not yield a symmetric equilibrium, while the Tit-for-Tat automaton does.

When the size of the automaton is not restricted to be less than k, the constant-defect equilibrium does exist. However, there is still a large class of machine games in which other equilibria exist in which some amount of cooperation occurs, as shown in the following result.

Theorem 6.1.8 *For any integer x, there exists an integer k_0 such that for all $k > k_0$, any machine game $G^M = (\{1, 2\}, \mathcal{M}, G)$ of the k-period repeated Prisoner's Dilemma game G, in which $k^{1/x} \leq \min\{S(\mathcal{M}_1), S(\mathcal{M}_2)\} \leq \max\{S(\mathcal{M}_1), S(\mathcal{M}_2)\} \leq k^x$ holds has a Nash equilibrium in which the average payoffs to each player are at least $3 - \frac{1}{x}$.*

Thus the average payoffs to each player can be much higher than $(1, 1)$; in fact they can be arbitrarily close to $(3, 3)$, depending on the choice of x. While this result uses pure strategies for both players, a stronger result can be proved through the use of a mixed-strategy equilibrium.

Theorem 6.1.9 *For every $\epsilon > 0$, there exists an integer k_0 such that for all $k > k_0$, any machine game $G^M = (\{1, 2\}, \mathcal{M}, G)$ of the k-period repeated Prisoner's Dilemma game G, in which $\min\{S(\mathcal{M}_1), S(\mathcal{M}_2)\} < 2^{\frac{\epsilon k}{12(1+\epsilon)}}$ has a Nash equilibrium in which the average payoffs to each player are at least $3 - \epsilon$.*

Thus, if even one of the players' automata has a size that is less than exponential in the length of the game, an equilibrium with some degree of cooperation exists.

Automata with a cost of complexity

Now, instead of imposing constraints on the complexity of the automata, we will incorporate this complexity as a cost into the agent's utility function. This could reflect, for example, the implementation cost of a strategy or the cost to learn it. While we cannot show theorems similar to those in the preceding section, it turns out that we can get mileage out of this idea even when we incorporate it in a minimal way. Specifically, an agent's disutility for complexity will only play a tie-breaking role.

lexicographic disutility for complexity

Definition 6.1.10 (Lexicographic disutility for complexity) *Agents have* lexicographic disutility for complexity *in a machine game if their utility functions $U_i(\cdot)$ in the induced normal-form game are replaced by preference orderings $\succeq_i$ such that $(M_1, M_2) \succ_i (M_1', M_2')$ whenever either $U_i(M_1, M_2) > U_i(M_1', M_2')$ or $U_i(M_1, M_2) = U_i(M_1', M_2')$ and $s(M_i) < s(M_i')$.*

Consider a machine game G^M of the discounted infinitely repeated Prisoner's Dilemma in which both players have a lexicographic disutility for complexity. The trigger strategy is an equilibrium strategy in the infinitely repeated Prisoner's Dilemma game with discounting. When the discount factor β is large enough, if player 2 is using the trigger strategy, then player 1 cannot achieve a higher payoff

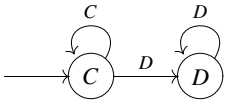

Figure 6.6 An automaton representing the trigger strategy.

by using any strategy other than the trigger strategy himself. We can represent the trigger strategy using the machine M shown in Figure 6.6. However, while no other machine can give player 1 a higher payoff, there does exist another machine that achieves the *same* payoff and is less complex. Player 1's machine M never enters the state D during play; it is designed only as a threat to the other player. Thus the machine which contains only the state C will achieve the same payoff as the machine M, but with less complexity. As a result, the outcome (M, M) is not a Nash equilibrium of the machine game G^M when agents have a lexicographic disutility for complexity.

We can also show several interesting properties of the equilibria of machine games in which agents have a lexicographic disutility for complexity. First, because machines in equilibrium must minimize complexity, they have no unused states. Thus we know that in an infinite game, every state must be visited in some period. Second, the strategies represented by the machines in a Nash equilibrium of the machine game also form a Nash equilibrium of the infinitely repeated game.

Computing best-response automata

In the previous sections we limited the rationality of agents in repeated games by bounding the number of states that they can use to represent their strategies. However, it could be the case that the number of states used by the equilibrium strategies is small, but the time required to compute them is prohibitively large. Furthermore, one can argue (by introspection, for example) that bounding the computation of an agent is a more appropriate means of capturing bounded rationality than bounding the number of states.

It seems reasonable that an equilibrium must be at least verifiable by agents. But this does not appear to be the case for finite automata. (The results that follow are for the limit-average case, but can be adapted to the discounted case as well.)

Theorem 6.1.11 *Given a two-player machine game $G^M = (N, \mathcal{M}, G)$ of a limit average infinitely repeated two-player game $G = (N, A, u)$ with unknown N, and a choice of automata $M_1, \ldots, M_n$ for all players, there does not exist a polynomial time algorithm for verifying whether M_i is a best-response automaton for player i.*

The news is not all bad; if we hold N fixed, then the problem does belong to P. We can explain this informally by noting that player i does not have to scan all of his possible strategies in order to decide whether automaton M_i is the best response; since he knows the strategies of the other players, he merely needs to scan the actual path taken on the game tree, which is bounded by the length of the game tree.

Notice that the previous result held even when the other players were assumed to play pure strategies. The following result shows that the verification problem is hard even in the two-player case when the players can randomize over machines.

Theorem 6.1.12 *Given a two-player machine game $G^M = (\{1, 2\}, \mathcal{M}, G)$ of a limit-average infinitely repeated game $G = (\{1, 2\}, A, u)$, and a mixed strategy for player 2 in which the set of automata that are played with positive probability is finite, the problem of verifying that an automaton M_1 is a best-response automaton for player 1 is NP-complete.*

So far we have abandoned the bounds on the number of states in the automata, and one might wonder whether such bounds could improve the worst-case complexity. However, for the repeated Prisoner's Dilemma game, it has the opposite effect: limiting the size of the automata under consideration increases the complexity of computing a best response. By Theorem 6.1.11 we know that when the size of the automata under consideration are unbounded and the number of agents is two, the problem of computing the best response is in the class P. The following result shows that when the automata under consideration are instead bounded, the problem becomes NP-complete.

Theorem 6.1.13 *Given a machine game $G^M = (\{1, 2\}, \mathcal{M}, G)$ of the limit average infinitely repeated Prisoner's Dilemma game G, an automaton M_2, and an integer k, the problem of computing a best-response automaton M_1 for player 1, such that $s(M_1) \leq k$, is NP-complete.*

From finite automata to Turing machines

Turing machines are more powerful than finite-state automata due to their infinite memories. One might expect that in this richer model, unlike with finite automata, game-theoretic results will be preserved. But they are not. For example, there is strong evidence (if not yet proof) that a Prisoner's Dilemma game of two Turing machines can have equilibria that are arbitrarily close to the repeated C payoff. Thus cooperative play can be approximated in equilibrium even if the machines memorize the entire history of the game and are capable of counting the number of repetitions.

The problem of computing a best response yields another unintuitive result. Even if we restrict the opponent to strategies for which the best-response Turing machine is computable, the general problem of finding the best response for any such input is not Turing computable when the discount factor is sufficiently close to one.

Theorem 6.1.14 *For the discounted, infinitely-repeated Prisoner's Dilemma game G, there exists a discount factor $\underline{\beta} > 0$ such that for any rational discount factor $\beta \in (\underline{\beta}, 1)$ there is no Turing-computable procedure for computing a best response to a strategy drawn from the set of all computable strategies that admit a computable best response.*

Finally, even before worrying about computing a best response, there is a more basic challenge: the best response to a Turing machine may not be a Turing machine!

Theorem 6.1.15 *For the discounted, infinitely-repeated Prisoner's Dilemma game G, there exists a discount factor $\underline{\beta} > 0$ such that for any rational discount factor $\beta \in (\underline{\beta}, 1)$ there exists an equilibrium profile (s_1, s_2) such that s_2 can be implemented by a Turing machine, but no best response to s_2 can be implemented by a Turing machine.*

6.2 Stochastic games

Intuitively speaking, a stochastic game is a collection of normal-form games; the agents repeatedly play games from this collection, and the particular game played at any given iteration depends probabilistically on the previous game played and on the actions taken by all agents in that game.

6.2.1 *Definition*

Stochastic games are very broad framework, generalizing both Markov decision processes (MDPs; see Appendix C) and repeated games. An MDP is simply a stochastic game with only one player, while a repeated game is a stochastic game in which there is only one stage game.

stochastic game

Markov game

Definition 6.2.1 (Stochastic game) *A* stochastic game *(also known as a* Markov game*) is a tuple (Q, N, A, P, R), where:*

- *Q is a finite set of games;*
- *N is a finite set of n players;*
- *$A = A_1 \times \cdots \times A_n$, where A_i is a finite set of actions available to player i;*
- *$P : Q \times A \times Q \mapsto [0, 1]$ is the transition probability function; $P(q, a, \hat{q})$ is the probability of transitioning from state q to state $\hat{q}$ after action profile a; and*
- *$R = r_1, \ldots, r_n$, where $r_i : Q \times A \mapsto \mathbb{R}$ is a real-valued payoff function for player i.*

In this definition we have assumed that the strategy space of the agents is the same in all games, and thus that the difference between the games is only in the payoff function. Removing this assumption adds notation, but otherwise presents no major difficulty or insights. Restricting Q and each A_i to be finite is a substantive restriction, but we do so for a reason; the infinite case raises a number of complications that we wish to avoid.

We have specified the payoff of a player at each stage game (or in each state), but not how these payoffs are aggregated into an overall payoff. To solve this problem, we can use solutions already discussed earlier in connection with infinitely repeated games (Section 6.1.2). Specifically, the two most commonly used aggregation methods are *average reward* and *future discounted reward*.

6.2.2 *Strategies and equilibria*

We now define the strategy space of an agent. Let $h_t = (q^0, a^0, q^1, a^1, \ldots, a^{t-1}, q^t)$ denote a history of t stages of a stochastic game, and let H_t be the set of all possible histories of this length. The set of deterministic strategies is the Cartesian product $\prod_{t,H_t} A_i$, which requires a choice for each possible history at each point in time. As in the previous game forms, an agent's strategy can consist of any mixture over deterministic strategies. However, there are several restricted classes of strategies that are of interest, and they form the following hierarchy. The first restriction is the requirement that the mixing take place at each history independently; this is the restriction to behavioral strategies seen in connection with extensive-form games.

Definition 6.2.2 (Behavioral strategy) *A behavioral strategy $s_i(h_t, a_{i_j})$ returns the probability of playing action a_{i_j} for history h_t.*

A Markov strategy further restricts a behavioral strategy so that, for a given time t, the distribution over actions depends only on the current state.

Markov strategy

Definition 6.2.3 (Markov strategy) *A* Markov strategy *s_i is a behavioral strategy in which $s_i(h_t, a_{i_j}) = s_i(h'_t, a_{i_j})$ if $q_t = q'_t$, where q_t and q'_t are the final states of h_t and h'_t, respectively.*

The final restriction is to remove the possible dependence on the time t.

stationary strategy

Definition 6.2.4 (Stationary strategy) *A* stationary strategy *s_i is a Markov strategy in which $s_i(h_{t_1}, a_{i_j}) = s_i(h'_{t_2}, a_{i_j})$ if $q_{t_1} = q'_{t_2}$, where q_{t_1} and q'_{t_2} are the final states of h_{t_1} and h'_{t_2}, respectively.*

Now we can consider the equilibria of stochastic games, a topic that turns out to be fraught with subtleties. The discounted-reward case is the less problematic one. In this case it can be shown that a Nash equilibrium exists in every stochastic game. In fact, we can state a stronger property. A strategy profile is called a *Markov perfect equilibrium* if it consists of only Markov strategies, and is a Nash equilibrium regardless of the starting state. In a sense, MPE plays a role analogous to the subgame-perfect equilibrium in perfect-information games.

Markov perfect equilibrium (MPE)

Theorem 6.2.5 *Every n-player, general-sum, discounted-reward stochastic game has a Markov perfect equilibrium.*

The case of average rewards presents greater challenges. For one thing, the limit average may not exist (i.e., although the stage-game payoffs are bounded, their average may cycle and not converge). However, there is a class of stochastic games that is well behaved in this regard. This is the class of *irreducible* stochastic games. A stochastic game is irreducible if every strategy profile gives rise to an irreducible Markov chain over the set of games, meaning that every game can be reached with positive probability regardless of the strategy adopted. In such games the limit averages are well defined, and we have the following theorem.

irreducible stochastic game

Theorem 6.2.6 *Every two-player, general-sum, average reward, irreducible stochastic game has a Nash equilibrium.*

Indeed, under the same condition we can state a folk theorem similar to that presented for repeated games in Section 6.1.2. That is, as long as we give each player an expected payoff that is at least as large as his minmax value, any feasible payoff pair can be achieved in equilibrium through the use of threats.

Theorem 6.2.7 *For every two-player, general-sum, irreducible stochastic game, and every feasible outcome with a payoff vector r that provides to each player at least his minmax value, there exists a Nash equilibrium with a payoff vector r. This is true for games with average rewards, as well as games with large enough discount factors (or, with players that are sufficiently patient).*

6.2.3 *Computing equilibria*

The algorithms and results for stochastic games depend greatly on whether we use discounted reward or average reward for the agent utility function. We will discuss both separately, starting with the discounted reward case. The first question to ask about the problem of finding a Nash equilibrium is whether a polynomial procedure is available. The fact that there exists an linear programming formulation for solving MDPs (for both the discounted reward and average reward cases) gives us a reason for optimism, since stochastic games are a generalization of MDPs. While such a formulation does not exist for the full class of stochastic games, it does for several nontrivial subclasses.

One such subclass is the set of two-player, general-sum, discounted-reward stochastic games in which the transitions are determined by a single player. The *single-controller* condition is formally defined as follows.

single-controller stochastic game

Definition 6.2.8 (Single-controller stochastic game) *A stochastic game is* single-controller *if there exists a player i such that $\forall q, q' \in Q, \forall a \in A$, $P(q, a, q') = P(q, a', q')$ if $a_i = a'_i$.*

The same results hold when we replace the single-controller restriction with the following pair of restrictions: that the state and action profile have independent effects on the reward achieved by each agent, and that the transition function only depends on the action profile. Formally, this pair is called the *separable reward state independent transition* condition.

Definition 6.2.9 (SR-SIT stochastic game) *A stochastic game is* separable reward state independent transition *(SR-SIT) if the following two conditions hold:*

- *there exist functions α, γ such that $\forall i, q \in Q, \forall \in A$ it is the case that $r_i(q, a) = \alpha(q) + \gamma(a)$; and*
- *$\forall q, q', q'' \in Q, \forall \in A$ it is the case that $P(q, a, q'') = P(q', a, q'')$.*

Even when the problem does not fall into one of these subclasses, practical solutions still exist for the discounted case. One such solution is to apply a modified version of Newton's method to a nonlinear program formulation of the problem. An advantage of this method is that no local minima exist. For zero-sum

games, an alternative is to use an algorithm developed by Shapley that is related to value iteration, a commonly-used method for solving MDPs (see Appendix C).

Moving on to the average reward case, we have to impose more restrictions in order to use a linear program than we did for the discounted reward case. Specifically, for the class of two-player, general-sum, average-reward stochastic games, the single-controller assumption no longer suffices—we also need the game to be zero sum.

Even when we cannot use a linear program, irreducibility allows us to use an algorithm that is guaranteed to converge. This algorithm is a combination of policy iteration (another method used for solving MDPs) and successive approximation.

6.3 Bayesian games

All of the game forms discussed so far assumed that all players know what game is being played. Specifically, the number of players, the actions available to each player, and the payoff associated with each action vector have all been assumed to be common knowledge among the players. Note that this is true even of imperfect-information games; the actual moves of agents are not common knowledge, but the game itself is. In contrast, *Bayesian games*, or games of incomplete information, allow us to represent players' uncertainties about the very game being played.[4] This uncertainty is represented as a probability distribution over a set of possible games. We make two assumptions.

Bayesian game

1. All possible games have the same number of agents and the same strategy space for each agent; they differ only in their payoffs.
2. The beliefs of the different agents are posteriors, obtained by conditioning a common prior on individual private signals.

The second assumption is substantive, and we return to it shortly. The first is not particularly restrictive, although at first it might seem to be. One can imagine many other potential types of uncertainty that players might have about the game—how many players are involved, what actions are available to each player, and perhaps other aspects of the situation. It might seem that we have severely limited the discussion by ruling these out. However, it turns out that these other types of uncertainty can be reduced to uncertainty only about payoffs via problem reformulation.

For example, imagine that we want to model a situation in which one player is uncertain about the number of actions available to the other players. We can reduce this uncertainty to uncertainty about payoffs by padding the game with irrelevant actions. For example, consider the following two-player game, in which the row player does not know whether his opponent has only the two strategies L and R or also the third one C:

4. It is easy to confuse the term "incomplete information" with "imperfect information"; don't. . .

	L	R
U	1, 1	1, 3
D	0, 5	1, 13

	L	C	R
U	1, 1	0, 2	1, 3
D	0, 5	2, 8	1, 13

Now consider replacing the leftmost, smaller game by a padded version, in which we add a new C column.

	L	C	R
U	1, 1	0, −100	1, 3
D	0, 5	2, −100	1, 13

Clearly the newly added column is dominated by the others and will not participate in any Nash equilibrium (or any other reasonable solution concept). Indeed, there is an isomorphism between Nash equilibria of the original game and the padded one. Thus the uncertainty about the strategy space can be reduced to uncertainty about payoffs.

Using similar tactics, it can be shown that it is also possible to reduce uncertainty about other aspects of the game to uncertainty about payoffs only. This is not a mathematical claim, since we have given no mathematical characterization of all the possible forms of uncertainty, but it is the case that such reductions have been shown for all the common forms of uncertainty.

common-prior assumption

The second assumption about Bayesian games is the *common-prior assumption*, addressed in more detail in our discussion of multiagent probabilities and KP-structures in Chapter 13. As discussed there, a Bayesian game thus defines not only the uncertainties of agents about the game being played, but also their beliefs about the beliefs of other agents about the game being played, and indeed an entire infinite hierarchy of nested beliefs (the so-called epistemic type space). As also discussed in Chapter 13, the common-prior assumption is a substantive assumption that limits the scope of applicability. We nonetheless make this assumption since it allows us to formulate the main ideas in Bayesian games, and without the assumption the subject matter becomes much more involved than is appropriate for this text. Indeed, most (but not all) work in game theory makes this assumption.

6.3.1 *Definition*

There are several ways of presenting Bayesian games; we will offer three different definitions. All three are equivalent, modulo some subtleties that lie outside the

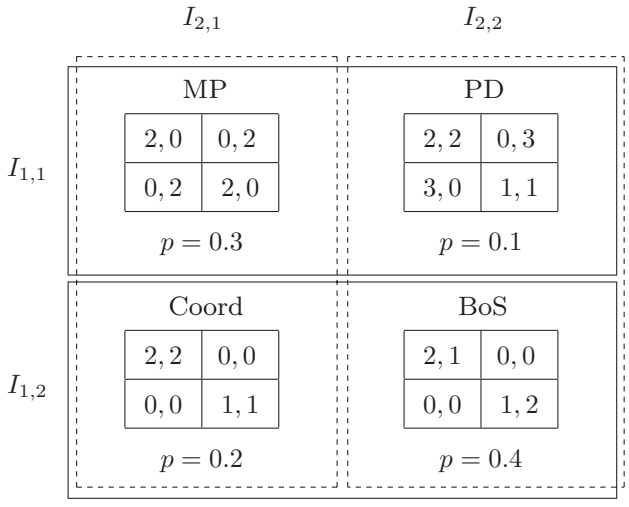

Figure 6.7 A Bayesian game.

scope of this book. We include all three since each formulation is useful in different settings and offers different intuition about the underlying structure of this family of games.

Information sets

First, we present a definition that is based on information sets. Under this definition, a Bayesian game consists of a set of games that differ only in their payoffs, a common prior defined over them, and a partition structure over the games for each agent.[5]

Bayesian game

Definition 6.3.1 (Bayesian game: information sets) *A* Bayesian game *is a tuple* (N, G, P, I) *where:*

- N *is a set of agents;*
- G *is a set of games with* N *agents each such that if* $g, g' \in G$ *then for each agent* $i \in N$ *the strategy space in* g *is identical to the strategy space in* g'*;*
- $P \in \Pi(G)$ *is a common prior over games, where* $\Pi(G)$ *is the set of all probability distributions over* G*; and*
- $I = (I_1, ..., I_N)$ *is a tuple of partitions of* G*, one for each agent.*

Figure 6.7 gives an example of a Bayesian game. It consists of four 2×2 games (Matching Pennies, Prisoner's Dilemma, Coordination and Battle of the Sexes), and each agent's partition consists of two equivalence classes.

Extensive form with chance moves

A second way of capturing the common prior is to hypothesize a special agent called Nature who makes probabilistic choices. While we could have Nature's

5. This combination of a common prior and a set of partitions over states of the world turns out to correspond to a KP-structure, defined in Chapter 13.

choice be interspersed arbitrarily with the agents' moves, without loss of generality we assume that Nature makes all its choices at the outset. Nature does not have a utility function (or, alternatively, can be viewed as having a constant one), and has the unique strategy of randomizing in a commonly known way. The agents receive individual signals about Nature's choice, and these are captured by their information sets in a standard way. The agents have no additional information; in particular, the information sets capture the fact that agents make their choices without knowing the choices of others. Thus, we have reduced games of incomplete information to games of imperfect information, albeit ones with chance moves. These chance moves of Nature require minor adjustments of existing definitions, replacing payoffs by their expectations given Nature's moves.[6]

For example, the Bayesian game of Figure 6.7 can be represented in extensive form as depicted in Figure 6.8.

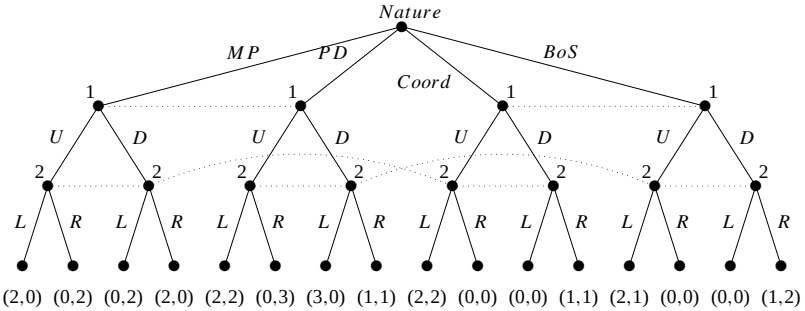

Figure 6.8 The Bayesian game from Figure 6.7 in extensive form.

Although this second definition of Bayesian games can be initially more intuitive than our first definition, it can also be more cumbersome to work with. This is because we use an extensive-form representation in a setting where players are unable to observe each others' moves. (Indeed, for the same reason we do not routinely use extensive-form games of imperfect information to model simultaneous interactions such as the Prisoner's Dilemma, though we could do so if we wished.) For this reason, we will not make further use of this definition. We close by noting one advantage that it does have, however: it extends very naturally to Bayesian games in which players move sequentially and do (at least sometimes) learn about previous players' moves.

Epistemic types

Recall that a game may be defined by a set of players, actions, and utility functions. In our first definition agents are uncertain about which game they are playing; however, each possible game has the same sets of actions and players, and so agents are really only uncertain about the game's utility function.

6. Note that the special structure of this extensive-form game means that we do not have to agonize over the refinements of Nash equilibrium; since agents have no information about prior choices made other than by Nature, all Nash equilibria are also sequential equilibria.

a_1	a_2	θ_1	θ_2	u_1	u_2
U	L	$\theta_{1,1}$	$\theta_{2,1}$	2	0
U	L	$\theta_{1,1}$	$\theta_{2,2}$	2	2
U	L	$\theta_{1,2}$	$\theta_{2,1}$	2	2
U	L	$\theta_{1,2}$	$\theta_{2,2}$	2	1
U	R	$\theta_{1,1}$	$\theta_{2,1}$	0	2
U	R	$\theta_{1,1}$	$\theta_{2,2}$	0	3
U	R	$\theta_{1,2}$	$\theta_{2,1}$	0	0
U	R	$\theta_{1,2}$	$\theta_{2,2}$	0	0

a_1	a_2	θ_1	θ_2	u_1	u_2
D	L	$\theta_{1,1}$	$\theta_{2,1}$	0	2
D	L	$\theta_{1,1}$	$\theta_{2,2}$	3	0
D	L	$\theta_{1,2}$	$\theta_{2,1}$	0	0
D	L	$\theta_{1,2}$	$\theta_{2,2}$	0	0
D	R	$\theta_{1,1}$	$\theta_{2,1}$	2	0
D	R	$\theta_{1,1}$	$\theta_{2,2}$	1	1
D	R	$\theta_{1,2}$	$\theta_{2,1}$	1	1
D	R	$\theta_{1,2}$	$\theta_{2,2}$	1	2

Figure 6.9 Utility functions u_1 and u_2 for the Bayesian game from Figure 6.7.

epistemic type

Our third definition uses the notion of an *epistemic type*, or simply a *type* as a way of defining uncertainty directly over a game's utility function.

Bayesian game

Definition 6.3.2 (Bayesian game: types) *A* Bayesian game *is a tuple* (N, A, Θ, p, u) *where:*

- N *is a set of agents;*
- $A = A_1 \times \cdots \times A_n$*, where* A_i *is the set of actions available to player* i*;*
- $\Theta = \Theta_1 \times \ldots \times \Theta_n$*, where* Θ_i *is the type space of player* i*;*
- $p : \Theta \mapsto [0, 1]$ *is a common prior over types; and*
- $u = (u_1, \ldots, u_n)$*, where* $u_i : A \times \Theta \mapsto \mathbb{R}$ *is the utility function for player* i*.*

The assumption is that all of the above is common knowledge among the players, and that each agent knows his own type. This definition can seem mysterious, because the notion of type can be rather opaque. In general, the type of agent encapsulates all the information possessed by the agent that is not common knowledge. This is often quite simple (e.g., the agent's knowledge of his private payoff function), but can also include his beliefs about other agents' payoffs, about their beliefs about his own payoff, and any other higher-order beliefs.

We can get further insight into the notion of a type by relating it to the formulation at the beginning of this section. Consider again the Bayesian game in Figure 6.7. For each of the agents we have two types, corresponding to his two information sets. Denote player 1's actions as U and D, player 2's actions as L and R. Call the types of the first agent $\theta_{1,1}$ and $\theta_{1,2}$, and those of the second agent $\theta_{2,1}$ and $\theta_{2,2}$. The joint distribution on these types is as follows: $p(\theta_{1,1}, \theta_{2,1}) = .3$, $p(\theta_{1,1}, \theta_{2,2}) = .1$, $p(\theta_{1,2}, \theta_{2,1}) = .2$, $p(\theta_{1,2}, \theta_{2,2}) = .4$. The conditional probabilities for the first player are $p(\theta_{2,1} \mid \theta_{1,1}) = 3/4$, $p(\theta_{2,2} \mid \theta_{1,1}) = 1/4$, $p(\theta_{2,1} \mid \theta_{1,2}) = 1/3$, and $p(\theta_{2,2} \mid \theta_{1,2}) = 2/3$. Both players' utility functions are given in Figure 6.9.

6.3.2 *Strategies and equilibria*

Now that we have defined Bayesian games, we must explain how to reason about them. We will do this using the epistemic type definition, because that is the

definition most commonly used in mechanism design (discussed in Chapter 10), one of the main applications of Bayesian games. All of the concepts defined below can also be expressed in terms of the first two Bayesian game definitions as well.

The first task is to define an agent's strategy space in a Bayesian game. Recall that in an imperfect-information extensive-form game a pure strategy is a mapping from information sets to actions. The definition is similar in Bayesian games: a pure strategy $\alpha_i : \Theta_i \mapsto A_i$ is a mapping from every type agent i could have to the action he would play if he had that type. We can then define mixed strategies in the natural way as probability distributions over pure strategies. As before, we denote a mixed strategy for i as $s_i \in S_i$, where S_i is the set of all i's mixed strategies. Furthermore, we use the notation $s_j(a_j|\theta_j)$ to denote the probability under mixed strategy s_j that agent j plays action a_j, given that j's type is θ_j.

Next, since we have defined an environment with multiple sources of uncertainty, we will pause to reconsider the definition of an agent's expected utility. In a Bayesian game setting, there are three meaningful notions of expected utility: *ex post*, *ex interim* and *ex ante*. The first is computed based on all agents' actual types, the second considers the setting in which an agent knows his own type but not the types of the other agents, and in the third case the agent does not know anybody's type.

ex post expected utility

Definition 6.3.3 (*Ex post* expected utility) *Agent i's* ex post *expected utility in a Bayesian game (N, A, Θ, p, u), where the agents' strategies are given by s and the agent' types are given by θ, is defined as*

$$EU_i(s, \theta) = \sum_{a \in A} \left(\prod_{j \in N} s_j(a_j|\theta_j) \right) u_i(a, \theta). \tag{6.1}$$

In this case, the only uncertainty concerns the other agents' mixed strategies, since agent i's *ex post* expected utility is computed based on the other agents' actual types. Of course, in a Bayesian game no agent *will* know the others' types; while that does not prevent us from offering the definition given, it might make the reader question its usefulness. We will see that this notion of expected utility is useful both for defining the other two and also for defining a specialized equilibrium concept.

Definition 6.3.4 (*Ex interim* expected utility) *Agent i's* ex interim *expected utility in a Bayesian game (N, A, Θ, p, u), where i's type is θ_i and where the agents' strategies are given by the mixed-strategy profile s, is defined as*

$$EU_i(s, \theta_i) = \sum_{\theta_{-i} \in \Theta_{-i}} p(\theta_{-i}|\theta_i) \sum_{a \in A} \left(\prod_{j \in N} s_j(a_j|\theta_j) \right) u_i(a, \theta_{-i}, \theta_i), \tag{6.2}$$

or equivalently as

$$EU_i(s, \theta_i) = \sum_{\theta_{-i} \in \Theta_{-i}} p(\theta_{-i}|\theta_i) EU_i(s, (\theta_i, \theta_{-i})). \tag{6.3}$$

Thus, i must consider every assignment of types to the other agents θ_{-i} and every pure action profile a in order to evaluate his utility function $u_i(a, \theta_i, \theta_{-i})$. He must weight this utility value by two amounts: the probability that the other players' types would be θ_{-i} given that his own type is θ_i, and the probability that the pure action profile a would be realized given all players' mixed strategies and types. (Observe that agents' types may be correlated.) Because uncertainty over mixed strategies was already handled in the *ex post* case, we can also write *ex interim* expected utility as a weighted sum of $EU_i(s, \theta)$ terms.

Finally, there is the *ex ante* case, where we compute i's expected utility under the joint mixed strategy s without observing any agents' types.

ex ante expected utility

Definition 6.3.5 (*Ex ante* expected utility) *Agent i's* ex ante *expected utility in a Bayesian game (N, A, Θ, p, u), where the agents' strategies are given by the mixed-strategy profile s, is defined as*

$$EU_i(s) = \sum_{\theta \in \Theta} p(\theta) \sum_{a \in A} \left(\prod_{j \in N} s_j(a_j|\theta_j) \right) u_i(a, \theta), \tag{6.4}$$

or equivalently as

$$EU_i(s) = \sum_{\theta \in \Theta} p(\theta) EU_i(s, \theta), \tag{6.5}$$

or again equivalently as

$$EU_i(s) = \sum_{\theta_i \in \Theta_i} p(\theta_i) EU_i(s, \theta_i). \tag{6.6}$$

Next, we define best response.

best response in a Bayesian game

Definition 6.3.6 (Best response in a Bayesian game) *The set of agent i's* best responses *to mixed-strategy profile s_{-i} are given by*

$$BR_i(s_{-i}) = \arg\max_{s_i' \in S_i} EU_i(s_i', s_{-i}). \tag{6.7}$$

Note that BR_i is a set because there may be many strategies for i that yield the same expected utility. It may seem odd that BR is calculated based on i's *ex ante* expected utility. However, write $EU_i(s)$ as $\sum_{\theta_i \in \Theta_i} p(\theta_i) EU_i(s, \theta_i)$ and observe that $EU_i(s_i', s_{-i}, \theta_i)$ does not depend on strategies that i would play if his type were not θ_i. Thus, we are in fact performing independent maximization of i's *ex interim* expected utilities conditioned on each type that he could have. Intuitively speaking, if a certain action is best after the signal is received, it is also the best conditional plan devised ahead of time for what to do should that signal be received.

We are now able to define the Bayes–Nash equilibrium.

Bayes–Nash equilibrium

Definition 6.3.7 (Bayes–Nash equilibrium) *A* Bayes–Nash equilibrium *is a mixed-strategy profile s that satisfies $\forall i \;\; s_i \in BR_i(s_{-i})$.*

This is exactly the definition we gave for the Nash equilibrium in Definition 3.3.4: each agent plays a best response to the strategies of the other players. The difference from Nash equilibrium, of course, is that the definition of Bayes–Nash equilibrium is built on top of the Bayesian game definitions of best response and expected utility. Observe that we would not be able to define equilibrium in this way if an agent's strategies were not defined for every possible type. In order for a given agent i to play a best response to the other agents $-i$, i must know what strategy each agent would play for each of his possible types. Without this information, it would be impossible to evaluate the term $EU_i(s_i', s_{-i})$ in Equation (6.7).

6.3.3 *Computing equilibria*

Despite its similarity to the Nash equilibrium, the Bayes–Nash equilibrium may seem more conceptually complicated. However, as we did with extensive-form games, we can construct a normal-form representation that corresponds to a given Bayesian game.

As with games in extensive form, the induced normal form for Bayesian games has an action for every pure strategy. That is, the actions for an agent i are the distinct mappings from Θ_i to A_i. Each agent i's payoff given a pure-strategy profile s is his *ex ante* expected utility under s. Then, as it turns out, the Bayes–Nash equilibria of a Bayesian game are precisely the Nash equilibria of its induced normal form. This fact allows us to note that Nash's theorem applies directly to Bayesian games, and hence that Bayes–Nash equilibria always exist.

An example will help. Consider the Bayesian game from Figure 6.9. Note that in this game each agent has four possible pure strategies (two types and two actions). Then player 1's four strategies in the Bayesian game can be labeled UU, UD, DU, and DD: UU means that 1 chooses U regardless of his type, UD that he chooses U when he has type $\theta_{1,1}$ and D when he has type $\theta_{1,2}$, and so forth. Similarly, we can denote the strategies of player 2 in the Bayesian game by RR, RL, LR, and LL.

We now define a 4×4 normal-form game in which these are the four strategies of the two agents, and the payoffs are the expected payoffs in the individual games, given the agents' common prior beliefs. For example, player 2's *ex ante* expected utility under the strategy profile (UU, LL) is calculated as follows:

$$\begin{aligned}
&u_2(UU, LL) \\
&\quad = \sum_{\theta \in \Theta} p(\theta) u_2(U, L, \theta) \\
&\quad = p(\theta_{1,1}, \theta_{2,1}) u_2(U, L, \theta_{1,1}, \theta_{2,1}) + p(\theta_{1,1}, \theta_{2,2}) u_2(U, L, \theta_{1,1}, \theta_{2,2}) \\
&\qquad + p(\theta_{1,2}, \theta_{2,1}) u_2(U, L, \theta_{1,2}, \theta_{2,1}) + p(\theta_{1,2}, \theta_{2,2}) u_2(U, L, \theta_{1,2}, \theta_{2,2}) \\
&\quad = 0.3(0) + 0.1(2) + 0.2(2) + 0.4(1) = 1.
\end{aligned}$$

Continuing in this manner, the complete payoff matrix can be constructed as shown in Figure 6.10.

	LL	LR	RL	RR
UU	2, 1	1, 0.7	1, 1.2	0, 0.9
UD	0.8, 0.2	1, 1.1	0.4, 1	0.6, 1.9
DU	1.5, 1.4	0.5, 1.1	1.7, 0.4	0.7, 0.1
DD	0.3, 0.6	0.5, 1.5	1.1, 0.2	1.3, 1.1

Figure 6.10 Induced normal form of the game from Figure 6.9.

Now the game may be analyzed straightforwardly. For example, we can determine that player 1's best response to RL is DU.

Given a particular signal, the agent can compute the posterior probabilities and recompute the expected utility of any given strategy vector. Thus in the previous example once the row agent gets the signal $\theta_{1,1}$ he can update the expected payoffs and compute the new game shown in Figure 6.11.

	LL	LR	RL	RR
UU	2, 0.5	1.5, 0.75	0.5, 2	0, 2.25
UD	2, 0.5	1.5, 0.75	0.5, 2	0, 2.25
DU	0.75, 1.5	0.25, 1.75	2.25, 0	1.75, 0.25
DD	0.75, 1.5	0.25, 1.75	2.25, 0	1.75, 0.25

Figure 6.11 *Ex interim* induced normal-form game, where player 1 observes type $\theta_{1,1}$.

Note that for the row player, DU is still a best response to RL; what has changed is how much better it is compared to the other three strategies. In particular, the row player's payoffs are now independent of his choice of which action to take upon observing type $\theta_{1,2}$; in effect, conditional on observing type $\theta_{1,1}$ the player needs only to select a single action U or D. (Thus, we could have written the *ex interim* induced normal form in Figure 6.11 as a table with four columns but only two rows.)

Although we can use this matrix to find best responses for player 1, it turns out to be meaningless to analyze the Nash equilibria in this payoff matrix. This is because these expected payoffs are not common knowledge; if the column player were to condition on his signal, he would arrive at a different set of numbers (though, again, for him best responses would be preserved). Ironically, it is only

in the induced normal form, in which the payoffs do not correspond to any *ex interim* assessment of any agent, that the Nash equilibria are meaningful.

Other computational techniques exist for Bayesian games that also have temporal structure—that is, for Bayesian games written using the "extensive form with chance moves" formulation, for which the game tree is smaller than its induced normal form. First, there is an algorithm for Bayesian games of perfect information that generalizes backward induction (defined in Section 5.1.4), is called *expectimax*. Intuitively, this algorithm is very much like the standard backward induction algorithm given in Figure 5.6. Like that algorithm, expectimax recursively explores a game tree, labeling each non-leaf node h with a payoff vector by examining the labels of each of h's child nodes—the actual payoffs when these child nodes are leaf nodes—and keeping the payoff vector in which the agent who moves at h achieves maximal utility. The new wrinkle is that chance nodes must also receive labels. Expectimax labels a chance node h with a weighted sum of the labels of its child nodes, where the weights are the probabilities that each child node will be selected. The same idea of labeling chance nodes with the expected value of the next node's label can also be applied to extend the minimax algorithm (from which expectimax gets its name) and alpha-beta pruning (see Figure 5.7) in order to solve zero-sum games. This is a popular algorithmic framework for building computer players for perfect-information games of chance such as Backgammon.

expectimax algorithm

There are also efficient computational techniques for computing sample equilibria of imperfect-information extensive-form games with chance nodes. In particular, all the computational results for computing with the sequence form that we discussed in Section 5.2.3 still hold when chance nodes are added. Intuitively, the only change we need to make is to replace our definition of the payoff function (Definition 5.2.7) with an expected payoff that supplies the expected value, ranging over Nature's possible actions, of the payoff the agent would achieve by following a given sequence. This means that we can sometimes achieve a substantial computational savings by working with the extensive-form representation of a Bayesian game, rather than considering the game's induced normal form.

6.3.4 Ex post *equilibrium*

Finally, working with *ex post* utilities allows us to define an equilibrium concept that is stronger than the Bayes–Nash equilibrium.

Definition 6.3.8 (*Ex post* equilibrium) *An* ex post *equilibrium is a mixed-strategy profile s that satisfies* $\forall \theta,\ \forall i, s_i \in \arg\max_{s_i' \in S_i} EU_i(s_i', s_{-i}, \theta)$.

ex post equilibrium

Observe that this definition does not presume that each agent actually *does* know the others' types; instead, it says that no agent would ever want to deviate from his mixed strategy *even if* he knew the complete type vector θ. This form of equilibrium is appealing because it is unaffected by perturbations in the type distribution $p(\theta)$. Said another way, an *ex post* equilibrium does not ever require any agent to believe that the others have accurate beliefs about his own type distribution. (Note that a standard Bayes–Nash equilibrium *can* imply this requirement.) The *ex post* equilibrium is thus similar in flavor to equilibria in

dominant strategies, which do not require agents to believe that other agents act rationally.

Indeed, many dominant strategy equilibria are also *ex post* equilibria, making it easy to believe that this relationship always holds. In fact, it does not, as the following example shows. Consider a two-player Bayesian game where each agent has two actions and two corresponding types ($\forall_{i \in N},\ A_i = \Theta_i = \{H, L\}$) distributed uniformly ($\forall_{i \in N},\ P(\theta_i = H) = 0.5$), and with the same utility function for each agent i:

$$u_i(a, \theta) = \begin{cases} 10 & a_i = \theta_{-i} = \theta_i; \\ 2 & a_i = \theta_{-i} \neq \theta_i; \\ 0 & \text{otherwise.} \end{cases}$$

In this game, each agent has a dominant strategy of choosing the action that corresponds to his type, $a_i = \theta_i$. An equilibrium in these dominant strategies is not *ex post* because if either agent knew the other's type, he would prefer to deviate to playing the strategy that corresponds to the other agent's type, $a_i = \theta_{-i}$.

Unfortunately, another sense in which *ex post* equilibria are in fact similar to equilibria in dominant strategies is that neither kind of equilibrium is guaranteed to exist.

Finally, we note that the term "*ex post* equilibrium" has been used in several different ways in the literature. One alternate usage requires that each agent's strategy constitute a best response not only to every possible *type* of the others, but also to every *pure strategy profile* that can be realized given the others' mixed strategies. (Indeed, this solution concept has also been applied in settings where there is no uncertainty about agents' types.) A third usage even more stringently requires that no agent ever play a mixed strategy. Both of these definitions can be useful, e.g., in the context of mechanism design (see Chapter 10). However, the advantage of Definition 6.3.8 is that of the three, it describes the most general prior-free equilibrium concept for Bayesian games.

6.4 Congestion games

Congestion games are a restricted class of games that are useful for modeling some important real-world settings and that also have attractive theoretical properties. Intuitively, they simplify the representation of a game by imposing constraints on the effects that a single agent's action can have on other agents' utilities.

6.4.1 *Definition*

Intuitively, in a congestion game each player chooses some subset from a set of resources, and the cost of each resource depends on the number of other agents who select it. Formally, a congestion game is single-shot n-player game, defined as follows.

congestion game

Definition 6.4.1 (Congestion game) *A* congestion game *is a tuple* (N, R, A, c), *where*

- N *is a set of* n *agents;*
- R *is a set of* r resources*;*

- $A = A_1 \times \cdots \times A_n$, *where* $A_i \subseteq 2^R \setminus \{\emptyset\}$ *is the set of* actions *for agent* i*; and*
- $c = (c_1, \ldots, c_r)$, *where* $c_k : \mathbb{N} \mapsto \mathbb{R}$ *is a* cost function *for resource* $k \in R$.

The players' utility functions are defined in terms of the cost functions c_k. Define $\# : R \times A \mapsto \mathbb{N}$ as a function that counts the number of players who took any action that involves resource r under action profile a. For each resource k, define a cost function $c_k : \mathbb{N} \mapsto \mathbb{R}$. Now we are ready to state the utility function,[7] which is the same for all players. Given a pure-strategy profile $a = (a_i, a_{-i})$,

$$u_i(a) = -\sum_{r \in R | r \in a_i} c_r(\#(r, a)).$$

Observe that while the agents can have different actions available to them, they all have the same utility function. Furthermore, observe that congestion games have an *anonymity* property: players care about *how many* others use a given resource, but they do not care about *which* others do so.

anonymity

One motivating example for this formulation is a computer network in which several users want to send a message from one node to another at approximately the same time. Each link connecting two nodes is a resource, and an action for a user is to select a path of links connecting their source and target node. The cost function for each resource expresses the latency on each link as a function of its congestion.

As the name suggests, a congestion game typically features functions $c_k(\cdot)$ that are increasing in the number of people who choose that resource, as would be the case in the network example. However, congestion games can just as easily handle positive externalities (or even cost functions that oscillate). A popular formulation that captures both types of externalities is the *Santa Fe (or, El Farol) Bar problem*, in which each of a set of people independently selects whether or not to go to the bar. The utility of attending increases with the number of other people who select the same night, up to the capacity of the bar. Beyond this point, utility decreases because the bar gets too crowded. Deciding not to attend yields a baseline utility that does not depend on the actions of the participants.[8]

Santa Fe Bar problem

6.4.2 *Computing equilibria*

Congestion games are interesting for reasons beyond the fact that they can compactly represent realistic n-player games like the examples given earlier. One particular example is the following result.

Theorem 6.4.2 *Every congestion game has a pure-strategy Nash equilibrium.*

We defer the proof for the moment, though we note that the property is important because mixed-strategy equilibria are open to criticisms that they are

7. This utility function is negated because the cost functions are historically understood as penalties that the agents want to minimize. We note that the c_r functions are also permitted to be negative.

8. Incidentally, this problem is typically studied in a repeated game context, in which (possibly boundedly rational) agents must learn to play an equilibrium. It is famous partly for not having a symmetric pure-strategy equilibrium, and has been generalized with the concept of *minority games*, in which agents get the highest payoff for choosing a minority action.

less likely than pure-strategy equilibria to arise in practice. Furthermore, this theorem tells us that if we want to compute a sample Nash equilibrium of a congestion game, we can look for a pure-strategy equilibrium. Consider the *myopic best-response* process, described in Figure 6.12.

myopic best-response

function MYOPICBESTRESPONSE (game G, action profile a) **returns** a
while *there exists an agent i for whom a_i is not a best response to a_{-i}* **do**
  $a_i' \leftarrow$ some best response by i to a_{-i}
  $a \leftarrow (a_i', a_{-i})$
return a

Figure 6.12 Myopic best response algorithm. It is invoked starting with an arbitrary (e.g., random) action profile a.

By the definition of equilibrium, MYOPICBESTRESPONSE returns a pure-strategy Nash equilibrium if it terminates. Because this procedure is so simple, it is an appealing way to search for an equilibrium. However, in general games MYOPICBESTRESPONSE can get caught in a cycle, even when a pure-strategy Nash equilibrium exists. For example, consider the game in Figure 6.13.

	L	C	R
U	$-1, 1$	$1, -1$	$-2, -2$
M	$1, -1$	$-1, 1$	$-2, -2$
D	$-2, -2$	$-2, -2$	$2, 2$

Figure 6.13 A game on which MYOPICBESTRESPONSE can fail to terminate.

This game has one pure-strategy Nash equilibrium, (D, R). However, if we run MYOPICBESTRESPONSE with $a = (L, U)$ the procedure will cycle forever. (Do you see why?) This suggests that MYOPICBESTRESPONSE may be too simplistic to be useful in practice. Interestingly, it *is* useful for congestion games.

Theorem 6.4.3 *The* MYOPICBESTRESPONSE *procedure is guaranteed to find a pure-strategy Nash equilibrium of a congestion game.*

6.4.3 *Potential games*

To prove the two theorems from the previous section, it is useful to introduce the concept of potential games.[9]

9. The potential games we discuss here are more formally known as *exact potential games*, though it is correct to shorten their name to the term *potential games*. There are other variants with somewhat different

potential game

Definition 6.4.4 (Potential game) *A game $G = (N, A, u)$ is a* potential game *if there exists a function $P : A \mapsto \mathbb{R}$ such that, for all $i \in N$, all $a_{-i} \in A_{-i}$ and $a_i, a_i' \in A_i$, $u_i(a_i, a_{-i}) - u_i(a_i', a_{-i}) = P(a_i, a_{-i}) - P(a_i', a_{-i})$.*

It is easy to prove the following property.

Theorem 6.4.5 *Every (finite) potential game has a pure-strategy Nash equilibrium.*

Proof. Let $a^* = \arg\max_{a \in A} P(a)$. Clearly for any other action profile a', $P(a^*) \geq P(a')$. Thus by the definition of a potential function, for any agent i who can change the action profile from a^* to a' by changing his own action, $u_i(a^*) \geq u_i(a')$. ■

Let $\mathbb{I}_{r \in a_i}$ be an indicator function that returns 1 if $r \in a_i$ for a given action a_i, and 0 otherwise. We also overload the notation # to give the expression $\#(r, a_{-i})$ its obvious meaning. Now we can show the following result.

Theorem 6.4.6 *Every congestion game is a potential game.*

Proof. We demonstrate that every congestion game has the potential function $P(a) = \sum_{r \in R} \sum_{j=1}^{\#(r,a)} c_r(j)$. To accomplish this, we must show that for any agent i and any action profiles (a_i, a_{-i}) and (a_i', a_{-i}), the difference between the potential function evaluated at these action profiles is the same as i's difference in utility.

$$
\begin{aligned}
& P(a_i, a_{-i}) - P(a_i', a_{-i}) \\
&= \left[\sum_{r \in R} \sum_{j=1}^{\#(r,(a_i,a_{-i}))} c_r(j) \right] - \left[\sum_{r \in R} \sum_{j=1}^{\#(r,(a_i',a_{-i}))} c_r(j) \right] \\
&= \left[\sum_{r \in R} \left(\left(\sum_{j=1}^{\#(r,(a_{-i}))} c_r(j) \right) + \mathbb{I}_{r \in a_i} c_r(j+1) \right) \right] \\
&\quad - \left[\sum_{r \in R} \left(\left(\sum_{j=1}^{\#(r,(a_{-i}))} c_r(j) \right) + \mathbb{I}_{r \in a_i'} c_r(j+1) \right) \right] \\
&= \left[\sum_{r \in R} \mathbb{I}_{r \in a_i} c_r(\#(r, a_{-i}) + 1) \right] - \left[\sum_{r \in R} \mathbb{I}_{r \in a_i'} c_r(\#(r, a_{-i}) + 1) \right] \\
&= \left[\sum_{r \in R | r \in a_i} c_r(\#(r, (a_i, a_{-i}))) \right] - \left[\sum_{r \in R | r \in a_i'} c_r(\#(r, (a_i', a_{-i}))) \right] \\
&= u_i(a_i, a_{-i}) - u_i(a_i', a_{-i})
\end{aligned}
$$

■

properties, such as *weighted potential games* and *ordinal potential games*. These variants differ in the expression that appears in Definition 6.4.4; for example, ordinal potential games generalize potential games with the condition $u_i(a_i, a_{-i}) - u_i(a_i', a_{-i}) > 0$ iff $P(a_i, a_{-i}) - P(a_i', a_{-i}) > 0$. More can be learned about these distinctions by consulting the reference given in the chapter notes; most importantly, potential games of all these variants are still guaranteed to have pure-strategy Nash equilibria.

Now that we have this result, the proof to Theorem 6.4.2 (stating that every congestion game has a pure-strategy Nash equilibrium) follows directly from Theorems 6.4.5 and 6.4.6. Furthermore, though we do not state this result formally, it turns out that the mapping given in Theorem 6.4.6 also holds in the other direction: every potential game can be represented as a congestion game.

Potential games (along with their equivalence to congestion games) also make it easy to prove Theorem 6.4.3 (stating that MyopicBestResponse will always find a pure-strategy Nash equilibrium), which we had previously deferred.

Proof of Theorem 6.4.3. By Theorem 6.4.6 it is sufficient to show that MyopicBestResponse finds a pure-strategy Nash equilibrium of any potential game. With every step of the while loop, $P(a)$ strictly increases, because by construction $u_i(a_i', a_{-i}) > u_(a_i, a_{-i})$, and thus by the definition of a potential function $P(a_i', a_{-i}) > P(a_i, a_{-i})$. Since there are only a finite number of action profiles, the algorithm must terminate. ■

Thus, when given a congestion game MyopicBestResponse will converge regardless of the cost functions (e.g., they do not need to be monotonic), the action profile with which the algorithm is initialized, and which agent we choose as agent i in the while loop (when there is more than one agent who is not playing a best response). Furthermore, we can see from the proof that it is not even necessary that agents *best respond* at every step. The algorithm will still converge to a pure-strategy Nash equilibrium by the same argument as long as agents deviate to a *better* response. On the other hand, it has recently been shown that the problem of finding a pure Nash equilibrium in a congestion game is PLS-complete: as hard to find as any other object whose existence is guaranteed by a potential function argument. Intuitively, this means that our problem is as hard as finding a local minimum in a traveling salesman problem using local search. This cautions us to expect that MyopicBestResponse will be inefficient in the worst case.

6.4.4 *Nonatomic congestion games*

A nonatomic congestion game is a congestion game that is played by an uncountably infinite number of players. These games are used to model congestion scenarios in which the number of agents is very large, and each agent's effect on the level of congestion is very small. For example, consider modeling traffic congestion in a freeway system.

nonatomic congestion games

Definition 6.4.7 (Nonatomic congestion game) *A* nonatomic congestion game *is a tuple* (N, μ, R, A, ρ, c)*, where:*

- $N = \{1, \ldots, n\}$ *is a set of* types *of players;*
- $\mu = (\mu_1, \ldots, \mu_n)$*; for each* $i \in N$ *there is a continuum of players represented by the interval* $[0, \mu_i]$*;*
- R *is a set of* k resources*;*
- $A = A_1 \times \cdots \times A_n$*, where* $A_i \subseteq 2^R \setminus \{\emptyset\}$ *is the set of* actions *for agents of type* i*;*

- $\rho = (\rho_1, \ldots, \rho_n)$, *where for each* $i \in N$, $\rho_i : A_i \times R \mapsto \mathbb{R}_+$ *denotes the amount of congestion contributed to a given resource* $r \in R$ *by players of type* i *selecting a given action* $a_i \in A_i$*; and*
- $c = (c_1, \ldots, c_k)$, *where* $c_r : \mathbb{R}_+ \mapsto \mathbb{R}$ *is a* cost function *for resource* $r \in R$*, and* c_r *is nonnegative, continuous and nondecreasing.*

To simplify notation, assume that $A_1, \ldots, A_n$ are disjoint; denote their union as $\mathcal{A}$. Let $\mathcal{S} = {\mathbb{R}_+}^{|\mathcal{A}|}$. An *action distribution* $s \in \mathcal{S}$ indicates how many players choose each action; by $s(a_i)$, denote the element of s that corresponds to the measure of the set of players of type i who select action $a_i \in A_i$. An action distribution s must have the properties that all entries are nonnegative real numbers and that $\sum_{a_i \in A_i} s(a_i) = \mu_i$. Note that $\rho_i(a_i, r) = 0$ when $r \notin a_i$. Overloading notation, we write as s_r the amount of congestion induced on resource $r \in R$ by action distribution s:

$$s_r = \sum_{i \in N} \sum_{a_i \in A_i} \rho_i(a_i, r) s(a_i).$$

We can now express the utility function. As in (atomic) congestion games, all agents have the same utility function, and the function depends only on how many agents choose each action rather than on these agents' identities. By $c_{a_i,s}$ we denote the cost, under an action distribution s, to agents of type i who choose action a_i. Then

$$c_{a_i}(s) = \sum_{r \in a_i} \rho(a_i, r) c_r(s_r),$$

and so we have $u_i(a_i, s) = -c_{a_i}(s)$. Finally, we can define the *social cost* of an action profile as the total cost born by all the agents,

$$C(s) = \sum_{i \in N} \sum_{a_i \in A_i} s(a_i) c_{a_i}(s).$$

Despite the fact that we have an uncountably infinite number of agents, we can still define a Nash equilibrium in the usual way.

Definition 6.4.8 (**Pure-strategy Nash equilibrium of a nonatomic congestion game**) *An action distribution* s *arises in a pure-strategy equilibrium of a nonatomic congestion game if for each player type* $i \in N$ *and each pair of actions* $a_1, a_2 \in A_i$ *with* $s(a_1) > 0$, $u_i(a_1, s) \geq u_i(a_2, s)$ *(and hence* $c_{a_1}(s) \leq c_{a_2}(s)$*).*

A couple of warnings are in order. First, the attentive reader will have noticed that we have glossed over the difference between actions and strategies. This is to simplify notation, and because we will only be concerned with pure-strategy equilibria. We do note that results exist concerning mixed-strategy equilibria of nonatomic congestion games; see the references cited at the end of the chapter. Second, we say only that an action distribution *arises in* an equilibrium because an action distribution does not identify the action taken by every individual agent, and hence cannot *constitute* an equilibrium. Nevertheless, from this point on we will ignore these issues.

We can now state some properties of nonatomic congestion games.

Theorem 6.4.9 *Every nonatomic congestion game has a pure-strategy Nash equilibrium.*

Furthermore, limiting ourselves by considering only pure-strategy equilibria is in some sense not restrictive.

Theorem 6.4.10 *All equilibria of a nonatomic congestion game have equal social cost.*

Intuitively, because the players are nonatomic, any mixed-strategy equilibrium corresponds to an "equivalent" pure-strategy equilibrium where the number of agents playing a given action is the expected number under the original equilibrium.

6.4.5 *Selfish routing and the price of anarchy*

selfish routing

Selfish routing is a model of how self-interested agents would route traffic through a congested network. This model was studied as early as 1920—long before game theory developed as a field. Today, we can understand these problems as nonatomic congestion games.

Defining selfish routing

First, let us formally define the problem. Let $G = (V, E)$ be a directed graph having n source–sink pairs $(s_1, t_1), \ldots, (s_n, t_n)$. Some volume of traffic must be routed from each source to each sink. For a given source–sink pair (s_i, t_i) let $\mathcal{P}_i$ denote the set of simple paths from s_i to t_i. We assume that $\mathcal{P} \neq \emptyset$ for all i; it is permitted for there to be multiple "parallel" edges between the same pair of nodes in V, and for paths from $\mathcal{P}_i$ and $\mathcal{P}_j$ $(j \neq i)$ to share edges. Let $\mu \in \mathbb{R}^n_+$ denote a vector of *traffic rates*; μ_i denotes the amount of traffic that must be routed from s_i to t_i. Finally, every edge $e \in E$ is associated with a cost function $c_e : \mathbb{R}_+ \mapsto \mathbb{R}$ (think of it an amount of delay) that can depend on the amount of traffic carried by the edge. The problem in selfish routing is to determine how the given traffic rates will lead traffic to flow along each edge, assuming that agents are selfish and will direct their traffic to minimize the sum of their own costs.

Selfish routing problems can be encoded as nonatomic congestion games as follows:

- N is the set of source–sink pairs;
- μ is the set of traffic rates;
- R is the set of edges E;
- A_i is the set of paths $\mathcal{P}_i$ from s_i to t_i;
- ρ_i is always 1; and
- c_r is the edge cost function c_e.

The price of anarchy

From the above reduction to nonatomic congestion games and from Theorems 6.4.9 and 6.4.10 we can conclude that every selfish routing problem has

at least one pure-strategy Nash equilibrium,[10] and that all of a selfish routing problem's equilibria have equal social cost. These properties allow us to ask an interesting question: how similar is the optimal social cost to the social cost under an equilibrium action distribution?

price of anarchy

Definition 6.4.11 (Price of anarchy) *The* price of anarchy *of a nonatomic congestion game* (N, μ, R, A, ρ, c) *having equilibrium* s *and social cost minimizing action distribution* s^* *is defined as* $\frac{C(s)}{C(s^*)}$ *unless* $C(s^*) = 0$*, in which case the price of anarchy is defined to be* 1.

Intuitively, the price of anarchy is the proportion of additional social cost that is incurred because of agents' self-interested behavior. When this ratio is close to 1 for a selfish routing problem, one can conclude that the agents are routing traffic about as well as possible, given the traffic rates and network structure. When this ratio is large, however, the agents' selfish behavior is causing significantly suboptimal network performance. In this latter case one might want to seek ways of changing either the network or the agents' behavior in order to reduce the social cost.

To gain a better understanding of the price of anarchy, and to lay the groundwork for some theoretical results, consider the examples in Figure 6.14.

Figure 6.14 Pigou's example: a selfish routing problem with an interesting price of anarchy. Left: linear version; right: nonlinear version.

In this example there is only one type of agent ($n = 1$) and the rate of traffic is 1 ($\mu_1 = 1$). There are two paths from s to t, one of which is relatively slow but immune to congestion, and the other of which has congestion-dependent cost.

Consider first the linear version of the problem given in Figure 6.14 (left). It is not hard to see that the Nash equilibrium is for all agents to choose the lower edge—indeed, this is a Nash equilibrium in dominant strategies. The social cost of this Nash equilibrium is 1. Consider what would happen if we required half of the agents to choose the upper edge, and the other half of the agents to choose the lower edge. In this case the social cost would be $3/4$, because half the agents would continue to pay a cost of 1, while half the agents would now pay a cost of only $1/2$. It is easy to show that this is the smallest social cost that can be achieved in this example, meaning that the price of anarchy here is $4/3$.

Now consider the nonlinear problem given in Figure 6.14 (right), where p is some large value. Again in the Nash equilibrium all agents will choose the lower edge, and again the social cost of this equilibrium is 1. Social cost is minimized when the marginal costs of the two edges are equalized; this occurs when a $(p+1)^{-1/p}$ fraction of the agents choose the lower edge. In this case the social cost is $1 - p \cdot (p+1)^{-(p+1)/p}$, which approaches 0 as $p \to \infty$. Thus we can

10. In the selfish routing literature these equilibria are known as Wardrop equilibria, after the author who first proposed their use. For consistency we avoid that term here.

see that the price of anarchy tends to infinity in the nonlinear version of Pigou's example as p grows.

Bounding the price of anarchy

These examples illustrate that the price of anarchy is unbounded for unrestricted cost functions. On the other hand, it turns out to be possible to offer bounds in the case where cost functions are restricted to a particular set $\mathcal{C}$. First, we must define the so-called Pigou bound:

$$\alpha(\mathcal{C}) = \sup_{c \in C} \sup_{x,\mu \geq 0} \frac{r \cdot c(r)}{x \cdot c(x) + (r - x)c(r)}.$$

When $\alpha(\mathcal{C})$ evaluates to $\frac{0}{0}$, we define it to be 1. We can now state a surprisingly strong result.

Theorem 6.4.12 *The price of anarchy of a selfish routing problem whose cost functions are taken from the set $\mathcal{C}$ is never more than $\alpha(\mathcal{C})$.*

Observe that Theorem 6.4.12 makes a very broad statement, bounding a selfish routing problem's price of anarchy regardless of network structure and for any given family of cost functions. Because α appears difficult to evaluate, one might find it hard to get excited about this result. However, α can be evaluated for a variety of interesting sets of cost functions. For example, when $\mathcal{C}$ is the set of linear functions $ax + b$ with $a, b \geq 0$, $\alpha(\mathcal{C}) = 4/3$. Indeed, $\alpha(\mathcal{C})$ takes the same value when $\mathcal{C}$ is the set of all convex functions. This means that the bound from Theorem 6.4.12 is tight for this set of functions: Pigou's linear example from Figure 6.14 (left) uses only convex cost functions and we have already shown that this problem has a price of anarchy of precisely $4/3$. The linear version of Pigou's example thus serves as a worst case for the price of anarchy among all selfish routing problems with convex cost functions. Because the price of anarchy is relatively close to 1 for networks with convex edge costs, this result indicates that centralized control of traffic offers limited benefit in this case.

What about other families of cost functions, such as polynomials with nonnegative coefficients and bounded degree? It turns out that the Pigou bound is also tight for this family and that the nonlinear variant of Pigou's example offers the worst-possible price of anarchy in this case (where p is the bound on the polynomials' degree). For this family $\alpha(\mathcal{C}) = [1 - p \cdot (p + 1)^{-(p+1)/p}]^{-1}$. To give some examples, this means that the price of anarchy is about 1.6 for $p = 2$, about 2.2 for $p = 4$, about 18 for $p = 100$ and—as it was earlier—unbounded as $p \to \infty$.

Results also exist bounding the price of anarchy for general nonatomic congestion games. It is beyond the scope of this section to state these results formally, but we note that they are qualitatively similar to the results given above. More information can be found in the references cited in the chapter notes.

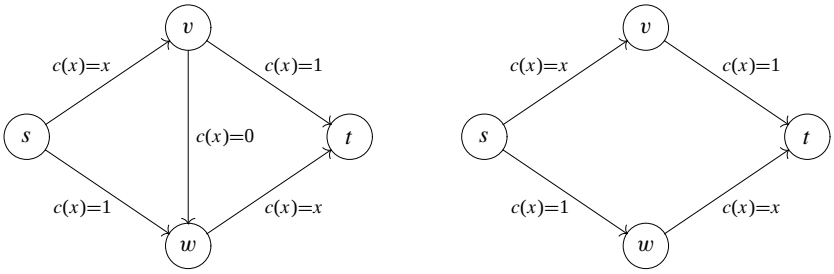

Figure 6.15 Braess' paradox: removing an edge that has zero cost can improve social welfare. Left: original network; right: after edge removal.

Reducing the social cost of selfish routing

When the equilibrium social cost is undesirably high, a network operator might want to intervene in some way in order to reduce it. First, we give an example to show that such interventions are possible, known as *Braess' paradox*.

Braess' paradox

Consider first the example in Figure 6.15 (Left). This selfish routing problem is essentially a more complicated version of the linear version of Pigou's example from Figure 6.14 (left). Again $n = 1$ and $\mu_1 = 1$. Agents have a weakly dominant strategy of choosing the path s-v-w-t, and so in equilibrium all traffic will flow along this path. The social cost in equilibrium is therefore 1. Minimal social cost is achieved by having half of the agents choose the path s-v-t and having the other half of the agents choose the path s-w-t; the social cost in this case is $3/4$. Like the linear version of Pigou's example, therefore, the price of anarchy is $4/3$.

The interesting thing about this new example is the role played by the edge v-w. One might intuitively believe that zero-cost edges can only help in routing problems, because they provide agents with a costless way of routing traffic from one node to another. At worst, one might reason, such edges would be ignored. However, this intuition is wrong. Consider the network in Figure 6.15 (right). This network was constructed from the network in Figure 6.15 (left) by removing the zero-cost edge v-w. In this modified problem agents no longer have a dominant strategy; the equilibrium is for half of them to choose each path. This is also the optimal action distribution, and hence the price of anarchy in this case is 1. We can now see the (apparent) paradox: removing even a zero-cost edge can transform a selfish routing problem from the very worst (a network having the highest price of anarchy possible given its family of cost functions) to the very best (a network in which selfish agents will choose to route themselves optimally).

A network operator facing a high price of anarchy might therefore want to remove one or more edges in order to improve the network's social cost in equilibrium. Unfortunately, however, the problem of determining which edges to remove is computationally hard.

Theorem 6.4.13 *It is NP-complete to determine whether there exists any set of edges whose removal from a selfish routing problem would reduce the social cost in equilibrium.*

In particular, this result implies that identifying the optimal set of edges to remove from a selfish routing problem in order to minimize the social cost in equilibrium is also NP-complete.

Of course, it is always possible to reduce a network's social cost in equilibrium by reducing all of the edge costs. (This could be done in an electronic network, for example, by installing faster routers.) Interestingly, even in the case where the edge functions are unconstrained and the price of anarchy is therefore unbounded, a relatively modest reduction in edge costs can outperform the imposition of centralized control in the original network.

Theorem 6.4.14 *Let Γ be a selfish routing problem, and let Γ' be identical to Γ except that each edge cost $c_e(x)$ is replaced by $c'_e(x) = c_e(x/2)/2$. The social cost in equilibrium of Γ' is less than or equal to the* optimal *social cost in Γ.*

This result suggests that when it is relatively inexpensive to speed up a network, doing so can have more significant benefits than getting agents to change their behavior.

Stackelberg routing

Finally, we will briefly mention two other methods of reducing social cost in equilibrium. First, in so-called *Stackelberg routing* a small fraction of agents are routed centrally, and the remaining population of agents is free to choose their own actions. It should already be apparent from the example in Figure 6.14 (right) that such an approach can be very effective in certain networks. Second, taxes can be imposed on certain edges in the graph in order to encourage agents to adopt more socially beneficial behavior. The dominant idea here is to charge agents according to "marginal cost pricing"—each agent pays the amount his presence cost other agents who are using the same edge.[11] Under certain assumptions taxes can be set up in a way that induces optimal action distributions; however, the taxes themselves can be very large. Various papers in the literature elaborate on and refine both of these ideas.

6.5 Computationally motivated compact representations

So far we have examined game representations that are motivated primarily by the goals of capturing relevant details of real-world domains and of showing that all games expressible in the representation share useful theoretical properties. Many of these representations—especially the normal and extensive forms—suffer from the problem that their encodings of interesting games are so large as to be impractical. For example, when you describe to someone the rules of poker, you do not give them a normal or extensive-form description; such a description would fill volumes and be almost unintelligible. Instead, you describe the rules of the game in a very compact form, which is possible because of the inherent structure of the game. In this section we explore some computationally motivated alternative representations that allow certain large games to be compactly described and also make it possible to efficiently find an equilibrium. The first two representations, graphical games and action-graph games, apply to normal-form games, while the following two, multiagent influence diagrams and the GALA language, apply to extensive-form games.

11. Here we anticipate the idea of *mechanism design*, introduced in Chapter 10, and especially the VCG mechanism from Section 10.4.

6.5.1 *The expected utility problem*

We begin by defining a problem that is fundamental to the discussion of computationally motivated compact representations.

ExpectedUtility problem

Definition 6.5.1 (ExpectedUtility) *Given a game (possibly represented in a compact form), a mixed-strategy profile s, and $i \in N$, the* ExpectedUtility *problem is to compute $EU_i(s)$, the expected utility of player i under mixed-strategy profile s.*

Our chief interest in this section will be in the computational complexity of the ExpectedUtility problem for different game representations. When we considered normal-form games, we showed (in Definition 3.2.7) that ExpectedUtility can be computed as

$$EU_i(s) = \sum_{a \in A} u_i(a) \prod_{j=1}^{n} s_j(a_j). \tag{6.8}$$

If we interpret Equation (6.8) as a simple algorithm, we have a way of solving ExpectedUtility in time exponential in the number of agents. This algorithm is exponential because, assuming for simplicity that all agents have the same number of actions, the size of A is $|A_i|^n$. However, since the representation size of a normal-form game is itself exponential in the number of agents (it is $O(|A_i|^n)$), the problem can in fact be solved in time linear in the size of the representation. Thus ExpectedUtility does not appear to be very computationally difficult.

Interestingly though, as game representations become exponentially more compact than the normal form, it grows more challenging to solve the ExpectedUtility problem efficiently. This is because our simple algorithm given by Equation (6.8) requires time exponential in the size of such more compact representations. The trick with compact representations, therefore, will not be simply finding some way of representing payoffs compactly—indeed, there are any number of schemes from the compression literature that could achieve this goal. Rather, we will want the additional property that the compactness of the representation can be leveraged by an efficient algorithm for computing ExpectedUtility.

The first challenge is to ensure that the inputs to ExpectedUtility can be specified compactly.

polynomial type

Definition 6.5.2 (Polynomial type) *A game representation has* polynomial type *if the number of agents n and the sizes of the action sets $|A_i|$ are polynomially bounded in the size of the representation.*

Representations always have polynomial type when their action sets are specified explicitly. However, some representations—such as the extensive form—implicitly specify action spaces that are exponential in the size of the representation and so do not have polynomial type.

When we combine the polynomial type requirement with a further requirement about ExpectedUtility being efficiently computable, we obtain the following theorem.

Theorem 6.5.3 *If a game representation satisfies the following properties:*

1. *the representation has polynomial type; and*
2. ExpectedUtility *can be computed using an arithmetic binary circuit consisting of a polynomial number of nodes, where each node evaluates to a constant value or performs addition, subtraction or multiplication on its inputs;*

then the problem of finding a Nash equilibrium in this representation can be reduced to the problem of finding a Nash equilibrium in a two-player normal-form game that is only polynomially larger.

We know from Theorem 4.2.1 in Section 4.2 that the problem of finding a Nash equilibrium in a two-player normal-form game is PPAD-complete. Therefore this theorem implies that if the above condition holds, the problem of finding a Nash equilibrium for a compact game representation is in PPAD. This should be understood as a positive result: if a game in its compact representation is exponentially smaller than its induced normal form, and if computing an equilibrium for this representation belongs to the same complexity class as computing an equilibrium of a normal-form game, then equilibria can be computed exponentially more quickly using the compact representation.

Observe that the second condition in Theorem 6.5.3 implies that the ExpectedUtility algorithm takes polynomial time; however, not every polynomial-time algorithm will satisfy this condition. Congestion games are an example of games that do meet the conditions of Theorem 6.5.3. We will see two more such representations in the next sections.

What about extensive-form games, which do not have polynomial type—might it be harder to compute their Nash equilibria? Luckily we can use behavioral strategies, which can be represented linearly in the size of the game tree. Then we obtain the following result.

Theorem 6.5.4 *The problem of computing a Nash equilibrium in behavioral strategies in an extensive-form game can be polynomially reduced to finding a Nash equilibrium in a two-player normal-form game.*

This shows that the speedups we achieved by using the sequence form in Section 5.2.3 were not achieved simply because of inefficiency in our algorithms for normal-form games. Instead, there is a fundamental computational benefit to working with extensive-form games, at least when we restrict ourselves to behavioral strategies.

Fast algorithms for solving ExpectedUtility are useful for more than just demonstrating the worst-case complexity of finding a Nash equilibrium for a game representation. ExpectedUtility is also a bottleneck step in several practical algorithms for computing Nash equilibria, such as the Govindan–Wilson algorithm or simplicial subdivision methods (see Section 4.3). Plugging a fast method for solving ExpectedUtility into one of these algorithms offers a simple way of more quickly computing a Nash equilibrium of a compactly represented game.

The complexity of the ExpectedUtility problem is also relevant to the computation of solution concepts other than the Nash equilibrium.

Theorem 6.5.5 *If a game representation has polynomial type and has a polynomial algorithm for computing* ExpectedUtility*, then a correlated equilibrium can be computed in polynomial time.*

The attentive reader may recall that we have already showed (in Section 4.6) that correlated equilibria can be identified in polynomial time by solving a linear program (Equations (4.52)–(4.54)). Thus, Theorem 6.5.5 may not seem very interesting. The catch, as with expected utility, is that while this LP has size polynomial in size of the normal form, its size would be exponential in the size of many compact representations. Specifically, there is one variable in the linear program for each action profile, and so overall the linear program has size exponential in any representation for which the simple ExpectedUtility algorithm discussed earlier is inadequate. Indeed, in these cases even *representing* a correlated equilibrium using these probabilities of action profiles would be exponential. Theorem 6.5.5 is thus a much deeper result than it may first seem. Its proof begins by showing that there exists a correlated equilibrium of every compactly represented game that can be written as the mixture of a polynomial number of *product distributions*, where a product distribution is a joint probability distribution over action profiles arising from each player independently randomizing over his actions (i.e., adopting a mixed-strategy profile). Since the theorem requires that the game representation has polynomial type, each of these product distributions can be compactly represented. Thus a polynomial mixture of product distributions can also be represented polynomially. The rest of the proof appeals to linear programming duality and to properties of the ellipsoid algorithm.

6.5.2 *Graphical games*

Graphical games are a compact representation of normal-form games that use graphical models to capture the *payoff independence* structure of the game. Intuitively, a player's payoff matrix can be written compactly if his payoff is affected only by a subset of the other players.

Let us begin with an example, which we call the Road game. Consider n agents, each of whom has purchased a piece of land alongside a road. Each agent has to decide what to build on his land. His payoff depends on what he builds himself, what is built on the land to either side of his own, and what is built across the road. Intuitively, the payoff relationships in this situation can be understood using the graph shown in Figure 6.16, where each node represents an agent.

Now let us define the representation formally. First, we define a neighborhood relation on a graph: the set of nodes connected to a given node, plus the node itself.

Definition 6.5.6 (Neighborhood relation) *For a graph defined on a set of nodes N and edges E, for every $i \in N$ define the* neighborhood relation *$v : N \mapsto 2^N$ as $v(i) = \{i\} \cup \{j | (j, i) \in E\}$.*

neighborhood relation

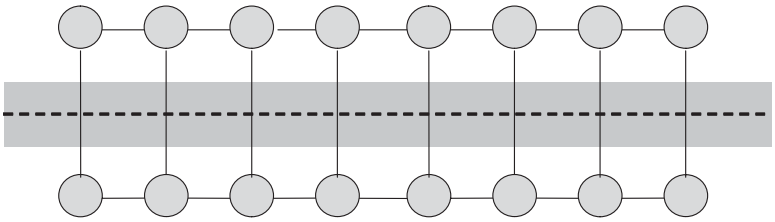

Figure 6.16 Graphical game representation of the Road game.

Now we can define the graphical game representation.

graphical game

Definition 6.5.7 (Graphical game) *A* graphical game *is a tuple* (N, E, A, u)*, where:*

- N *is a set of* n *vertices, representing agents;*
- E *is a set of undirected edges connecting the nodes* N*;*
- $A = A_1 \times \cdots \times A_n$*, where* A_i *is the set of actions available to agent* i*; and*
- $u = (u_1, \ldots, u_n)$, $u_i : A^{(i)} \mapsto \mathbb{R}$*, where* $A^{(i)} = \prod_{j \in \nu(i)} A_j$.

An edge between two vertices in the graph can be interpreted as meaning that the two agents are able to affect each other's payoffs. In other words, whenever two nodes i and j are *not* connected in the graph, agent i must always receive the same payoff under any action profiles (a_j, a_{-j}) and (a'_j, a_{-j}), $a_j, a'_j \in A_j$. Graphical games can represent any game, but of course they are not always compact. The space complexity of the representation is exponential in the size of the largest $\nu(i)$. In the example above the size of the largest $\nu(i)$ is 4, and this is independent of the total number of agents. As a result, the graphical game representation of the example requires space polynomial in n, while a normal-form representation would require space exponential in n.

The following is sufficient to show that the properties we discussed above in Section 6.5.1 hold for graphical games.

Lemma 6.5.8 *The* ExpectedUtility *problem can be computed in polynomial time for graphical games, and such an algorithm can be translated to an arithmetic circuit as required by Theorem 6.5.3.*

The way that graphical games capture payoff independence in games is similar to the way that Bayesian networks and Markov random fields capture conditional independence in multivariate probability distributions. It should therefore be unsurprising that many computations on graphical games can be performed efficiently using algorithms similar to those proposed in the graphical models literature. For example, when the graph (N, E) defines a tree, a message-passing algorithm called NashProp can compute an ϵ-Nash equilibrium in time polynomial in $1/\epsilon$ and the size of the representation. NashProp consists of two phases: a "downstream" pass in which messages are passed from the leaves to the root and then an "upstream" pass in which messages are passed from the root to the leaves. When the graph is a path, a similar algorithm can find an exact equilibrium in polynomial time.

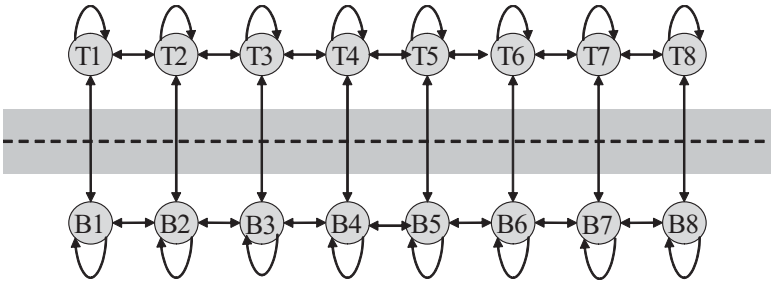

Figure 6.17 Modified Road game.

We may also be interested in finding pure-strategy Nash equilibria. Determining whether a pure-strategy equilibrium exists in a graphical game is NP-complete. However, the problem can be formulated as a constraint satisfaction problem (or alternatively as a Markov random field) and solved using standard algorithms. In particular, when the graph has constant *treewidth*,[12] the problem can be solved in polynomial time.

Graphical games have also been useful as a theoretical tool. For example, they are instrumental in the proof of Theorem 4.2.1, which showed that finding a sample Nash equilibrium of a normal-form game is PPAD-complete. Intuitively, graphical games are important to this proof because such games can be constructed to simulate arithmetic circuits in their equilibria.

6.5.3 *Action-graph games*

Consider a scenario similar to the Road game given in Section 6.5.2, the same or with one major difference: instead of deciding what to build, here agents need to decide *where* to build. Suppose each of the n agents is interested in opening a business (say a coffee shop), and can choose to locate in any block along either side of a road. Multiple agents can choose the same block. Agent i's payoff depends on the number of agents who chose the same block as he did, as well as the numbers of agents who chose each of the adjacent blocks of land. This game has an obvious graphical structure, which is illustrated in Figure 6.17. Here nodes correspond to actions, and each edge indicates that an agent who takes one action affects the payoffs of other agents who take some second action.

Notice that any pair of agents can potentially affect each other's payoffs by choosing the same or adjacent locations. This means that the graphical game representation of this game is a clique, and the space complexity of this representation is the same as that of the normal form (exponential in n). The problem is that graphical games are only compact for games with *strict payoff independencies*: that is, where there exist pairs of players who can never (directly) affect each other. This game exhibits *context-specific independence* instead: whether two agents

12. A graph's *treewidth* is a measure of how similar the graph is to a tree. It is defined using the *tree decomposition* of the graph. Many NP-complete problems on graphs can be solved efficiently when a graph has small treewidth.

are able to affect each other's payoffs depends on the actions they choose. The action-graph game (AGG) representation exploits this kind of context-specific independence. Intuitively, this representation is built around the graph structure shown in Figure 6.17. Since this graph has actions rather than agents serving as the nodes, it is referred to as an action graph.

action graph

Definition 6.5.9 (Action graph) *An* action graph *is a tuple* $(\mathcal{A}, E)$*, where* $\mathcal{A}$ *is a set of nodes corresponding to actions and* E *is a set of directed edges.*

We want to allow for settings where agents have different actions available to them, and hence where an agent's action set is not identical to $\mathcal{A}$. (For example, no two agents could be able to take the "same" action, or every agent could have the same action set as in Figure 6.17.) We thus define as usual a set of action profiles $A = A_1 \times \cdots \times A_n$, and then let $\mathcal{A} = \bigcup_{i \in N} A_i$. If two actions by different agents have the same name, they will collapse to the same element of $\mathcal{A}$; otherwise they will correspond to two different elements of $\mathcal{A}$.

Given an action graph and a set of agents, we can further define a *configuration*, which is a possible arrangement of agents over nodes in an action graph.

configuration (of an action-graph game)

Definition 6.5.10 (Configuration) *Given an action graph* $(\mathcal{A}, E)$ *and a set of action profiles* A*, a* configuration c *is a tuple of* $|\mathcal{A}|$ *nonnegative integers, where the* k^{th} *element* c_k *is interpreted as the number of agents who chose the* k^{th} *action* $\alpha_k \in \mathcal{A}$*, and where there exists some* $a \in A$ *that would give rise to* c*. Denote the set of all configurations as* C*.*

Observe that multiple action profiles might give rise to the same configuration, because configurations simply count the number of agents who took each action without worrying about which agent took which action. For example, in the Road game all action profiles in which exactly half of the agents take action $T1$ and exactly half the agents take action $B8$ give rise to the same configuration.

anonymity

Intuitively, configurations will allow AGGs to compactly represent *anonymity* structure: cases where an agent's payoffs depend on the aggregate behavior of other agents, but not on which particular agents take which actions. Recall that we saw such structure in congestion games (Section 6.4).

Intuitively, we will use the edges of the action graph to denote context-specific independence relations in the game. Just as we did with graphical games, we will define a utility function that depends on the actions taken in some local neighborhood. As it was for graphical games, the neighborhood ν will be defined by the edges E; indeed, we will use exactly the same definition (Definition 6.5.6). In action graph games the idea will be that the payoff of a player playing an action $\alpha \in \mathcal{A}$ only depends on the configuration over the neighbors of α.[13] We must therefore define notation for such a configuration over a neighborhood. Let $C^{(\alpha)}$ denote the set of all restrictions of configurations to the elements corresponding to the neighborhood of $\alpha \in \mathcal{A}$. (That is, each $c \in C^{(\alpha)}$ is a tuple of length $|\nu(\alpha)|$.) Then u_α, the utility for *any* agent who takes action $\alpha \in \mathcal{A}$, is a mapping from $C^{(\alpha)}$ to the real numbers.

13. We use the notation α rather than a to denote an element of $\mathcal{A}$ in order to emphasize that we speak about a single action rather than an action profile.

Summing up, we can state the formal definition of action-graph games as follows.

action-graph game (AGG)

Definition 6.5.11 *An* action-graph game (AGG) *is a tuple* $(N, A, (\mathcal{A}, E), u)$, *where:*

- N *is the set of agents;*
- $A = A_1 \times \cdots \times A_n$, *where* A_i *is the set of actions available to agent* i*;*
- $(\mathcal{A}, E)$ *is an action graph, where* $\mathcal{A} = \bigcup_{i \in N} A_i$ *is the set of distinct actions; and*
- $u = \{u_\alpha | \alpha \in \mathcal{A}\}$, $u_\alpha : C^{(\alpha)} \mapsto \mathbb{R}$.

Since each utility function is a mapping only from the possible configurations over the neighborhood of a given action, the utility function can be represented concisely. In the Road game, since each node has at most four incoming edges, we only need to store $O(n^4)$ numbers for each node, and $O(|\mathcal{A}|n^4)$ numbers for the entire game. In general, when the in-degree of the action graph is bounded by a constant, the space complexity of the AGG representation is polynomial in n.

Like graphical games, AGGs are fully expressive. Arbitrary normal-form games can be represented as AGGs with nonoverlapping action sets. Graphical games can be encoded in the same way, but with a sparser edge structure. Indeed, the AGG encoding of a graphical game is just as compact as the original graphical game.

Although it is somewhat involved to show why this is true, AGGs have the theoretical properties we have come to expect from a compact representation.

Theorem 6.5.12 *Given an AGG,* ExpectedUtility *can be computed in time polynomial in the size of the AGG representation by an algorithm represented as an arithmetic circuit as required by Theorem 6.5.3. In particular, if the in-degree of the action graph is bounded by a constant, the time complexity is polynomial in* n.

The AGG representation can be extended to include *function nodes*, which are special nodes in the action graph that do not correspond to actions. For each function node p, c_p is defined as a deterministic function of the configuration of its neighbors $\nu(p)$. Function nodes can be used to represent a utility function's intermediate parameters, allowing the compact representation of games with additional forms of independence structure. Computationally, when a game with function nodes has the property that each player affects the configuration c independently, ExpectedUtility can still be computed in polynomial time. AGGs can also be extended to exploit additivity in players' utility functions. Given both of these extensions, AGGs are able to compactly represent a broad array of realistic games, including congestion games.

6.5.4 *Multiagent influence diagrams*

multiagent influence diagrams (MAIDs)

Multiagent influence diagrams (MAIDs) are a generalization of *influence diagrams* (IDs), a compact representation for decision-theoretic reasoning in the single-agent case. Intuitively, MAIDs can be seen as a combination of graphical

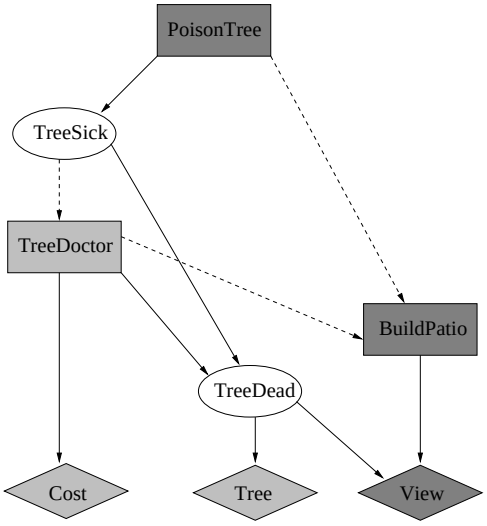

Figure 6.18 A multiagent influence diagram. Nodes for Alice are in dark gray, while Bob's are in light gray.

games and extensive-form games with chance moves (see Section 6.3). Not all variables (moves by nature) and action nodes depend on all other variables and action nodes, and only the dependencies need to be represented and reasoned about.

We will give a brief overview of MAIDs using the following example. Alice is considering building a patio behind her house, and the patio would be more valuable to her if she could get a clear view of the ocean. Unfortunately, there is a tree in her neighbor Bob's yard that blocks her view. Being somewhat unscrupulous, Alice considers poisoning Bob's tree, which might cause it to become sick. Bob cannot tell whether Alice has poisoned his tree, but he can tell if the tree is getting sick, and he has the option of calling in a tree doctor (at some cost). The attention of a tree doctor reduces the chance that the tree will die during the coming winter. Meanwhile, Alice must make a decision about building her patio before the weather gets too cold. When she makes this decision, she knows whether a tree doctor has come, but she cannot observe the health of the tree directly. A MAID for this scenario is shown in Figure 6.18.

Chance variables are represented as ovals, decision variables as rectangles, and utility variables as diamonds. Each variable has a set of parents, which may be chance variables or decision variables. Each chance node is characterized by a conditional probability distribution, which defines a distribution over the variable's domain for each possible instantiation of its parents. Similarly, each utility node records the conditional value for the corresponding agent. If multiple utility nodes exist for the same agent, as they do for in this example for Bob, the total utility is simply the sum of the values from each node. Decision variables differ in that their parents (connected by dotted arrows) are the variables that an agent observes when making his decision. This allows us to represent the information sets in a compact way.

For each decision node, the corresponding agent constructs a decision rule, which is a distribution over the domain of the decision variable for each possible

instantiation of this node's parents. A strategy for an agent consists of a decision rule for each of his decision nodes. Since a decision node acts as a chance node once its decision rule is set, we can calculate the expected utility of an agent given a strategy profile. As you would expect, a strategy profile is a Nash equilibrium in a MAID if no agent can improve its expected utility by switching to a different set of decision rules.

This example shows several of the advantages of the MAID representation over the equivalent extensive-form game representation. Since there are a total of five chance and decision nodes and all variables are binary, the game tree would have 32 leaves, each with a value for both agents. In the MAID, however, we only need four values for each agent to fill tables for the utility nodes. Similarly, redundant chance nodes of the game tree are replaced by small conditional probability tables. In general, the space savings of MAIDs can be exponential (although it is possible that this relationship is reversed if the game tree is sufficiently asymmetric).

The most important advantage of MAIDs is that they allow more efficient algorithms for computing equilibria, as we will informally show for the example.

strategic relevance

The efficiency of the algorithm comes from exploiting the property of *strategic relevance* in a way that is related to backward induction in perfect-information games. A decision node D_2 is strategically relevant to another decision node D_1 if, to optimize the rule at D_1, the agent needs to consider the rule at D_2. We omit a formal definition of strategic relevance, but point out that it can be computed in polynomial time.

No decision nodes are strategically relevant to *BuildPatio* for Alice, because she observes both of the decision nodes (*PoisonTree* and *TreeDoctor*) that could affect her utility before she has to make this decision. Thus, when finding an equilibrium, we can optimize this decision rule independently of the others and effectively convert it into a chance node. Next, we observe that *PoisonTree* is not strategically relevant to *TreeDoctor*, because any influence that *PoisonTree* has on a utility node for Bob must go through *TreeSick*, which is a parent of *TreeDoctor*. After optimizing this decision node, we can obviously optimize *PoisonTree* by itself, yielding an equilibrium strategy profile.

Obviously not all games allow such a convenient decomposition. However, as long as there exists some subset of the decision nodes such that no node outside of this subset is relevant to any node in the subset, then we can achieve some computational savings by jointly optimizing the decision rules for this subset before tackling the rest of the problem. Using this general idea, an equilibrium can often be found exponentially more quickly than in standard extensive-form games.

An efficient algorithm also exists for computing ExpectedUtility for MAIDs.

Theorem 6.5.13 *The* ExpectedUtility *problem for MAIDs can be computed in time polynomial in the size of the MAID representation.*

Unfortunately the only known algorithm for efficiently solving ExpectedUtility in MAIDS uses division and so cannot be directly translated to an arithmetic circuit as required in Theorem 6.5.3, which does not allow division

```
game(blind_tic_tac_toe,                                          (1)
  [ players : [a,b],                                             (2)
    objects : [grid_board : array('$size', '$size')],            (3)
    params : [size],                                             (4)
    flow : (take_turns(mark,unless(full),until(win))),           (5)
    mark : (choose('$player', (X, Y, Mark),                      (6)
                 (empty(X,Y), member(Mark, [x,o]))),             (7)
              reveal('$opponent',(X,Y)),                         (8)
              place((X,Y),Mark)),                                (9)
    full : (\+(empty(_,_)) → outcome(draw)),                     (10)
    win : (straight_line(_,_,length = 3,                         (11)
             contains(Mark)) → outcome(wins('$player')))]).      (12)
```

Figure 6.19 A GALA description of Blind Tic-Tac-Toe.

operations. It is unknown whether the problem of finding a Nash equilibrium in a MAID can be reduced to finding a Nash equilibrium in a two-player game. Nevertheless many other applications for computing ExpectedUtility we discussed in Section 6.5.1 apply to MAIDs. For example, the ExpectedUtility algorithm can be used as a subroutine to Govindan and Wilson's algorithm for computing Nash equilibria in extensive-form games (see Section 4.3).

6.5.5 *GALA*

While MAIDs allow us to capture exactly the relevant information needed to make a decision at each point in the game, we still need to explicitly record each choice point of the game. When, instead of modeling real-world setting, we are modeling a board or card game, this task would be rather cumbersome, if not impossible. The key property of these games that is not being exploited is their repetitive nature—the game alternates between the opponents whose possible moves are independent of the depth of the game tree, and can instead be defined in terms of the current state of the game and an unchanging set of rules. The GALA Prolog-based language *GALA* exploits this fact to allow concise specifications of large, complex games.

We present the main ideas of the language using the code in Figure 6.19 for an imperfect-information variant of Tic-Tac-Toe. Each player can mark a square with either an "x" or an "o," but the opponent sees only the position of the mark, not its type. A player wins if his move creates a line of the same type of mark.

Lines 3 and 5 define the central components of the representation—the object `grid_board` that records all marks, and the flow of the game, which is defined as two players alternating moves until either the board is full or one of the them wins the game. Lines 6–12 then provide the definitions of the terms used in line 5. Three of the functions found in these lines are particularly important because of their relation to the corresponding extensive-form game: `choose` (line 8) defines the available actions at each node, `reveal` (line 6) determines the information sets of the players, and `outcome` (lines 10 and 12) defines the payoffs at the leaves.

Reading through the code in Figure 6.19, one finds not only primitives like `array`, but also several high-level modules, like `straight_line`, that are

not defined. The GALA language contains many such predicates, built up from primitives, that were added to handle conditions common to games people play. For example, the high-level predicate `straight_line` is defined using the intermediate-level predicate `chain`, which in turn is defined to take a predicate and a set as input and return true if the predicate holds for the entire set. The idea behind intermediate-level predicates is that they make it easier to define the high-level predicates specific to a game. For example, `chain` can be used in poker to define a flush.

On top of the language, the GALA system was implemented to take a description of a game in the GALA language, generate the corresponding game tree, and then solve the game using the sequence form of the (defined in Section 5.2.3).

Since we lose the space savings of the GALA language when we actually solve the game, the main advantage of the language is the ease with which it allows a human to describe a game to the program that will solve it.

6.6 History and references

Some of the earliest and most influential work on repeated games is Luce and Raiffa [1957a] and Aumann [1959]. Of particular note is that the former provided the main ideas behind the folk theorem and that the latter explored the theoretical differences between finitely and infinitely repeated games. Aumann's work on repeated games led to a Nobel Prize in 2005. Our proof of the folk theorem is based on Osborne and Rubinstein [1994]. For an extensive discussion of the Tit-for-Tat strategy in repeated Prisoner's Dilemma, and in particular this strategy's strong performance in a tournament of computer programs, see Axelrod [1984].

While most game theory textbooks have material on so-called bounded rationality, the most comprehensive repository of results in the area was assembled by Rubinstein [1998]. Some of the specific references are as follows. Theorem 6.1.8 is due to Neyman [1985], while Theorem 6.1.9 is due to Papadimitriou and Yannakakis [1994]. Theorem 6.1.11 is due to Gilboa [1988], and Theorem 6.1.12 is due to Ben-Porath [1990]. Theorem 6.1.13 is due to Papadimitriou [1992]. Finally, Theorems 6.1.14 and 6.1.15 are due to Nachbar and Zame [1996].

Stochastic games were introduced in Shapley [1953]. The state of the art regarding them circa 2003 appears in the edited collection Neyman and Sorin [2003]. Filar and Vrieze [1997] provide a rigorous introduction to the topic, integrating MDPs (or single-agent stochastic games) and two-person stochastic games.

Bayesian games were introduced by Harsanyi [1967–1968]; in 1994 he received a Nobel Prize, largely because of this work.

Congestion games were first defined by Rosenthal [1973]; later potential games were introduced by Monderer and Shapley [1996a] and were shown to be equivalent to congestion games (up to isomorphism). The PLS-completeness result is due to Fabrikant et al. [2004]. Nonatomic congestion games are due to Schmeidler [1973]. Selfish routing was first studied as early as 1920 [Pigou, 1920; Beckmann

et al., 1956]. Pigou's example comes from the former reference; Braess' paradox was introduced in Braess [1968]. The Wardrop equilibrium is due to Wardrop [1952]. The concept of the price of anarchy is due to Koutsoupias and Papadimitriou [1999]. Most of the results in Section 6.4.5 are due to Roughgarden and his coauthors; see his recent book Roughgarden [2005]. Similar results have also been shown for broader classes of nonatomic congestion games; see Roughgarden and Tardos [2004] and Correa et al. [2005].

Theorems 6.5.3 and 6.5.4 are due to Daskalakis et al. [2006a]. Theorem 6.5.5 is due to Papadimitriou [2005]. Graphical games were introduced in Kearns et al. [2001]. The problem of finding pure Nash equilibria in graphical games was analyzed in Gottlob et al. [2003] and Daskalakis and Papadimitriou [2006]. Action graph games were defined in Bhat and Leyton-Brown [2004] and extended in Jiang and Leyton-Brown [2006]. Multiagent influence diagrams were introduced in Koller and Milch [2003], which also contains the running example we used for that section. A related notion of *game networks* was concurrently developed by La Mura [2000]. Theorem 6.5.13 is due to Blum et al. [2006]. GALA is described in Koller and Pfeffer [1995], which also contained the sample code for the Tic-Tac-Toe example.

game network