26 State Uncertainty

The multiagent models discussed so far in this part of the book have assumed that all agents can observe the true state. Just as an MDP can be extended to include partial observability, so can an MG be extended to produce a *partially observable Markov game* (*POMG*).[1] In fact, a POMG generalizes all the other problems presented in this book. These complex problems can be used to represent domains in which multiple agents receive partial or noisy observations of the environment. This generality makes modeling and solving POMGs computationally challenging. This chapter defines the POMG, outlines policy representations, and presents solution methods.

26.1 Partially Observable Markov Games

A POMG (algorithm 26.1) can be seen as either an extension of MGs to partial observability or as an extension of POMDPs to multiple agents. Each agent $i \in \mathcal{I}$ selects an action $a^i \in \mathcal{A}^i$ based only on local observations o^i made of a shared state s. The true state of the system $s \in \mathcal{S}$ is shared by all agents, but it is not necessarily fully observed. The initial state is drawn from a known initial state distribution b. The likelihood of transitioning from state s to state s' under their joint action $\mathbf{a}$ follows $T(s' \mid s, \mathbf{a})$. A joint reward $\mathbf{r}$ is generated following $R^i(s, \mathbf{a})$, as in MGs. Each agent strives to maximize its own accumulated reward. After all agents perform their joint action $\mathbf{a}$, a *joint observation* is emitted by the environment $\mathbf{o} = (o^1, \ldots, o^k)$ from a *joint observation space* $\mathcal{O} = \mathcal{O}^1 \times \cdots \times \mathcal{O}^k$. Each agent then receives an individual observation o^i from this joint observation. The crying baby problem is extended to multiple agents in example 26.1.

In POMDPs, we were able to maintain a belief state, as discussed in chapter 19, but this approach is not possible in POMGs. Individual agents cannot perform

[1] A POMG is also called a *partially observable stochastic game* (*POSG*). POMGs are closely related to the extensive form game with imperfect information. H. Kuhn, "Extensive Games and the Problem of Information," in *Contributions to the Theory of Games II*, H. Kuhn and A. Tucker, eds., Princeton University Press, 1953, pp. 193–216. The model was later introduced to the artificial intelligence community. E. A. Hansen, D. S. Bernstein, and S. Zilberstein, "Dynamic Programming for Partially Observable Stochastic Games," in *AAAI Conference on Artificial Intelligence* (*AAAI*), 2004.

the same kind of belief updates as in POMDPs because the joint actions and joint observations are not observed. Inferring a probability distribution over joint actions requires that each agent reason about the other agents reasoning about each other, who are in turn reasoning about each other, and so on. Inferring a distribution over the other observations is just as complicated because the observations depend on the actions of the other agents.[2]

Because of the difficulty of explicitly modeling beliefs in POMGs, we will focus on policy representations that do not require a belief to determine an action. We can use the tree-based conditional plan representation and the graph-based controller representation introduced in the earlier chapters on POMDPs. As in MGs, each agent in a POMG acts according to a policy π^i, or equivalently, the agents act together according to a joint policy $\boldsymbol{\pi} = (\pi^1, \ldots, \pi^k)$.

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

Algorithm 26.1. Data structure for a POMG.

Consider a multiagent POMG generalization of the crying baby problem. We have two caregivers taking care of a baby. As in the POMDP version, the states are the baby being hungry or sated. Each caregiver's actions are to feed, sing, or ignore the baby. If both caregivers choose to perform the same action, the cost is halved. For example, if both caregivers feed the baby, then the reward is only -2.5 instead of -5. However, the caregivers do not perfectly observe the state of the baby. Instead, they rely on the noisy observations of the baby crying, both with the same observation. As a consequence of the reward structure, there is a trade-off between helping each other and greedily choosing a less costly action.

Example 26.1. The multicaregiver crying baby problem as a POMG. Appendix F.14 provides additional details.

[2] The *Interactive POMDP* (*I-POMDP*) model attempts to capture this infinite regression. P. J. Gmytrasiewicz and P. Doshi, "A Framework for Sequential Planning in Multi-Agent Settings," *Journal of Artificial Intelligence Research*, vol. 24, no. 1, pp. 49–79, 2005. While this is a computationally complex framework because it reasons in both time and depth, algorithms for such models have advanced tremendously toward the goal of pragmatic use cases. E. Sonu, Y. Chen, and P. Doshi, "Decision-Theoretic Planning Under Anonymity in Agent Populations," *Journal of Artificial Intelligence Research*, vol. 59, pp. 725–770, 2017.

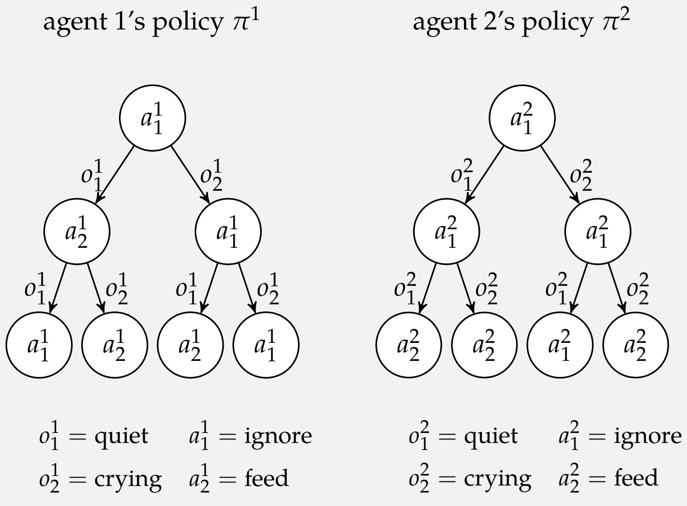
26.2 Policy Evaluation

This section discusses how to evaluate joint policies represented as either tree-based conditional plans or graph-based controllers. As in the context of POMDPs, we use conditional plans to represent deterministic policies and controllers to represent stochastic policies.

26.2.1 Evaluating Conditional Plans

Recall that a conditional plan (section 20.2) is a tree where actions are associated with nodes and observations are associated with edges. Each agent has its own tree and initially selects the action associated with its root. After making an observation, each agent proceeds down the tree, taking the edge associated with their observation. The process of taking actions and selecting edges based on observations continues until reaching the end of the tree. Example 26.2 shows a joint policy consisting of a conditional plan for each agent.

Here is a joint policy $\boldsymbol{\pi} = (\pi^1, \pi^2)$ represented as two-step conditional plans for the multicaregiver crying baby problem:



Example 26.2. A two-agent, two-step joint policy using conditional plans for the multicaregiver crying baby problem.

We can compute the joint utility function $\mathbf{U}^{\boldsymbol{\pi}}$ recursively, similar to what was done in equation (20.8) for POMDPs when starting in state s:

$$\mathbf{U}^{\boldsymbol{\pi}}(s) = \mathbf{R}(s, \boldsymbol{\pi}()) + \gamma \left[\sum_{s'} T(s' \mid s, \boldsymbol{\pi}()) \sum_{\mathbf{o}} O(\mathbf{o} \mid \boldsymbol{\pi}(), s') \mathbf{U}^{\boldsymbol{\pi}(\mathbf{o})}(s') \right] \tag{26.1}$$

where $\boldsymbol{\pi}()$ is the vector of actions at the root of the tree associated with $\boldsymbol{\pi}$ and $\boldsymbol{\pi}(\mathbf{o})$ is the vector of subplans associated with the various agents observing their components of the joint observation $\mathbf{o}$.

The utility associated with policy $\boldsymbol{\pi}$ from initial state distribution b is given by

$$\mathbf{U}^{\boldsymbol{\pi}}(b) = \sum_{s} b(s) \mathbf{U}^{\boldsymbol{\pi}}(s) \tag{26.2}$$

Algorithm 26.2 provides an implementation of this.

```
function lookahead(𝒫::POMG, U, s, a)
    𝒮, 𝒪, T, O, R, γ = 𝒫.𝒮, joint(𝒫.𝒪), 𝒫.T, 𝒫.O, 𝒫.R, 𝒫.γ
    u′ = sum(T(s,a,s′)*sum(O(a,s′,o)*U(o,s′) for o in 𝒪) for s′ in 𝒮)
    return R(s,a) + γ*u′
end

function evaluate_plan(𝒫::POMG, π, s)
    a = Tuple(πi() for πi in π)
    U(o,s′) = evaluate_plan(𝒫, [πi(oi) for (πi, oi) in zip(π,o)], s′)
    return isempty(first(π).subplans) ? 𝒫.R(s,a) : lookahead(𝒫, U, s, a)
end

function utility(𝒫::POMG, b, π)
    u = [evaluate_plan(𝒫, π, s) for s in 𝒫.𝒮]
    return sum(bs * us for (bs, us) in zip(b, u))
end
```

Algorithm 26.2. Conditional plans represent policies in a finite-horizon POMG. They are defined for a single agent in algorithm 20.1. We can compute the utility associated with executing a joint policy `π` represented by conditional plans when starting from a state `s`. Computing the utility from an initial state distribution `b` involves taking a weighted average of utilities when starting from different states.

26.2.2 *Evaluating Stochastic Controllers*

A controller (section 23.1) is represented as a stochastic graph. The controller associated with agent i is defined by the action distribution $\psi^i(a^i \mid x^i)$ and successor distribution $\eta^i(x^{i\prime} \mid x^i, a^i, o^i)$. The utility of being in state s with joint node $\mathbf{x}$ active and following joint policy $\boldsymbol{\pi}$ is

$$\mathbf{U}^{\boldsymbol{\pi}}(\mathbf{x}, s) = \sum_{\mathbf{a}} \prod_{i} \psi^i(a^i \mid x^i) \left(\mathbf{R}(s, \mathbf{a}) + \gamma \sum_{s'} T(s' \mid s, \mathbf{a}) \sum_{\mathbf{o}} O(\mathbf{o} \mid \mathbf{a}, s') \sum_{\mathbf{x}'} \prod_{i} \eta^i(x^{i\prime} \mid x^i, a^i, o^i) \mathbf{U}^{\boldsymbol{\pi}}(\mathbf{x}', s') \right) \tag{26.3}$$

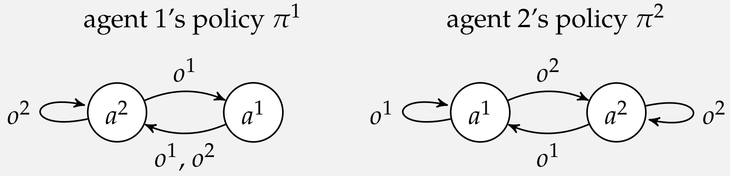
Policy evaluation in this context involves solving this system of linear equations. Alternatively, we can use iterative policy evaluation similar to algorithm 23.2 for POMDPs. The utility when starting from an initial state distribution b and joint controller state $\mathbf{x}$ is

$$\mathbf{U}^{\boldsymbol{\pi}}(\mathbf{x}, b) = \sum_s b(s)\mathbf{U}(\mathbf{x}, s) \tag{26.4}$$

Example 26.3 shows a joint stochastic controller.

Example 26.3. A two-agent joint policy using controllers for the multicaregiver crying baby problem.

Here is a joint controller policy $\boldsymbol{\pi} = (\pi^1, \pi^2)$ for the two caregivers in the crying baby problem. Each controller has two nodes, $X^i = \{x_1^i, x_2^i\}$:



$o^1 =$ quiet $\quad a^1 =$ ignore

$o^1 =$ hungry $\quad a^1 =$ feed

26.3 Nash Equilibrium

As with simple games and MGs, a *Nash equilibrium* for a POMG is when all agents act according to a best response policy to each other, such that no agents have an incentive to deviate from their policy. Nash equilibria for POMGs tend to be incredibly computationally difficult to solve. Algorithm 26.3 computes a d-step Nash equilibrium for a POMG. It enumerates all of its possible d-step joint conditional plans to construct a simple game, as shown in example 26.4. A Nash equilibrium for this simple game is also a Nash equilibrium for the POMG.

The simple game has the same agents as the POMG. There is a joint action in the simple game for every joint conditional plan in the POMG. The reward received for each action is equal to the utilities under the joint conditional plan in the POMG. A Nash equilibrium of this constructed simple game can directly be applied as a Nash equilibrium of the POMG.

```
struct POMGNashEquilibrium
    b # initial belief
    d # depth of conditional plans
end

function create_conditional_plans(𝒫, d)
    ℐ, 𝒜, 𝒪 = 𝒫.ℐ, 𝒫.𝒜, 𝒫.𝒪
    Π = [[ConditionalPlan(ai) for ai in 𝒜[i]] for i in ℐ]
    for t in 1:d
        Π = expand_conditional_plans(𝒫, Π)
    end
    return Π
end

function expand_conditional_plans(𝒫, Π)
    ℐ, 𝒜, 𝒪 = 𝒫.ℐ, 𝒫.𝒜, 𝒫.𝒪
    return [[ConditionalPlan(ai, Dict(oi => πi for oi in 𝒪[i]))
        for πi in Π[i] for ai in 𝒜[i]] for i in ℐ]
end

function solve(M::POMGNashEquilibrium, 𝒫::POMG)
    ℐ, γ, b, d = 𝒫.ℐ, 𝒫.γ, M.b, M.d
    Π = create_conditional_plans(𝒫, d)
    U = Dict(π => utility(𝒫, b, π) for π in joint(Π))
    𝒢 = SimpleGame(γ, ℐ, Π, π -> U[π])
    π = solve(NashEquilibrium(), 𝒢)
    return Tuple(argmax(πi.p) for πi in π)
end
```

Algorithm 26.3. A Nash equilibrium is computed for a POMG 𝒫 with initial state distribution `b` by creating a simple game of all conditional plans to some depth `d`. We solve for a Nash equilibrium in this simple game using algorithm 24.5. For simplicity, we select the most probable joint policy. Alternatively, we can randomly select the joint policy at the start of execution.

Example 26.4. Computing a Nash equilibrium for the multicaregiver crying baby problem by converting it into a simple game where the actions correspond to conditional plans.

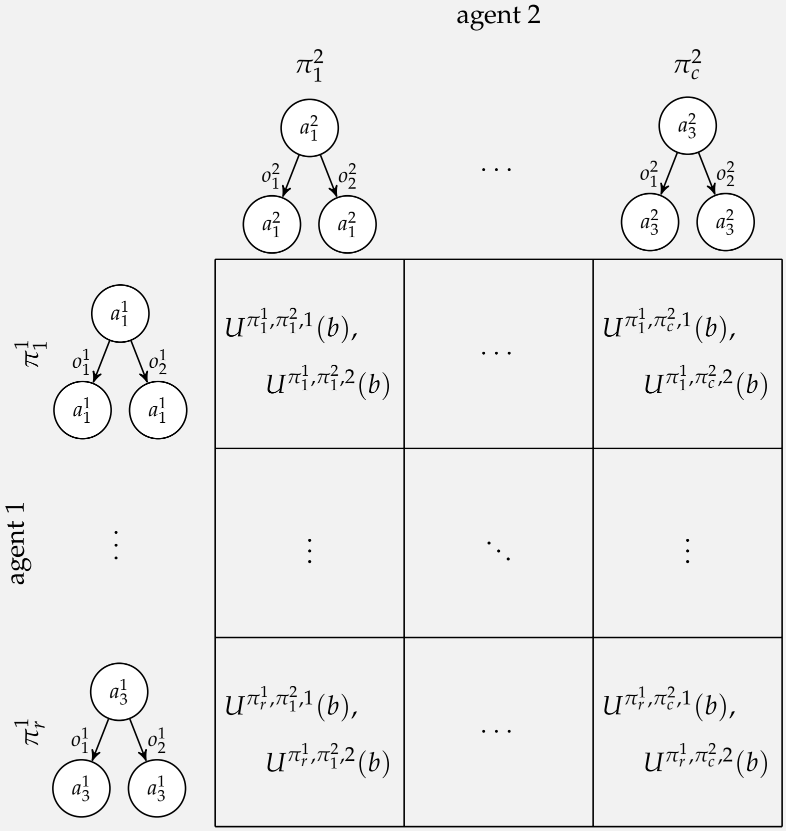
Consider the multicaregiver crying baby problem with a two-step horizon. Recall that for each agent i, there are three actions

$$\mathcal{A}^i = \{a_1^i, a_2^i, a_3^i\} = \{\text{feed}, \text{sing}, \text{ignore}\}$$

and two observations

$$\mathcal{O}^i = \{o_1^i, o_2^i\} = \{\text{cry}, \text{silent}\}$$

Converting this POMG to a simple game results in the following game table. Each caregiver selects simple game actions that correspond to a complete conditional plan. The simple game reward for each agent is the utility associated with the joint policy.



	π_1^2	$\cdots$	π_c^2
π_1^1	$U^{\pi_1^1,\pi_1^2,1}(b)$, $U^{\pi_1^1,\pi_1^2,2}(b)$	$\cdots$	$U^{\pi_1^1,\pi_c^2,1}(b)$, $U^{\pi_1^1,\pi_c^2,2}(b)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
π_r^1	$U^{\pi_r^1,\pi_1^2,1}(b)$, $U^{\pi_r^1,\pi_1^2,2}(b)$	$\cdots$	$U^{\pi_r^1,\pi_c^2,1}(b)$, $U^{\pi_r^1,\pi_c^2,2}(b)$

26.4 Dynamic Programming

The approach taken in the previous section for computing a Nash equilibrium is typically extremely computationally expensive because the actions correspond to all possible conditional plans to some depth. We can adapt the value iteration approach for POMDPs (section 20.5), where we iterated between expanding the depth of the set of considered conditional plans and pruning suboptimal plans. While the worst-case computational complexity is the same as that of the full expansion of all policy trees, this incremental approach can lead to significant savings.

Algorithm 26.4 implements this dynamic programming approach. It begins by constructing all one-step plans. We prune any plans that are dominated by another plan, and we then expand all combinations of one-step plans to produce two-step plans. This procedure of alternating between expansion and pruning is repeated until the desired horizon is reached.

The pruning step eliminates all dominated policies. A policy π^i belonging to an agent i can be pruned if there exists another policy $\pi^{i\prime}$ that always performs at least as well as π^i. Although computationally expensive, this condition can be checked by solving a linear program. This process is related to controller node pruning in POMDPs (algorithm 23.4).

It would be computationally intractable to solve a separate linear program for every possible combination of the other agent's policies $\boldsymbol{\pi}^{-i}$. Instead, we can take a much more efficient approach that will never prune an optimal policy but may not be able to prune all suboptimal policies. A policy π^i is dominated by $\pi^{i\prime}$ if there is no $b(\boldsymbol{\pi}^{-i}, s)$ between other joint policies $\boldsymbol{\pi}^{-i}$ and states s such that

$$\sum_{\boldsymbol{\pi}^{-i}} \sum_{s} b(\boldsymbol{\pi}^{-i}, s) U^{\pi^{i\prime}, \boldsymbol{\pi}^{-i}, i}(s) \geq \sum_{\boldsymbol{\pi}^{-i}} \sum_{s} b(\boldsymbol{\pi}^{-i}, s) U^{\pi^{i}, \boldsymbol{\pi}^{-i}, i}(s) \tag{26.5}$$

Here, b is a joint distribution over the policies of other agents and the state. As mentioned at the start of this chapter, it is generally infeasible to compute a belief state, but equation (26.5) checks the space of beliefs for individual policy domination.

We can construct a single linear program to check equation (26.5).[3] If the linear

[3] A similar linear program was created to prune alpha vectors in POMDPs in equation (20.16).

```
struct POMGDynamicProgramming
    b   # initial belief
    d   # depth of conditional plans
end

function solve(M::POMGDynamicProgramming, 𝒫::POMG)
    ℐ, 𝒮, 𝒜, R, γ, b, d = 𝒫.ℐ, 𝒫.𝒮, 𝒫.𝒜, 𝒫.R, 𝒫.γ, M.b, M.d
    Π = [[ConditionalPlan(ai) for ai in 𝒜[i]] for i in ℐ]
    for t in 1:d
        Π = expand_conditional_plans(𝒫, Π)
        prune_dominated!(Π, 𝒫)
    end
    𝒢 = SimpleGame(γ, ℐ, Π, π -> utility(𝒫, b, π))
    π = solve(NashEquilibrium(), 𝒢)
    return Tuple(argmax(πi.p) for πi in π)
end

function prune_dominated!(Π, 𝒫::POMG)
    done = false
    while !done
        done = true
        for i in shuffle(𝒫.ℐ)
            for πi in shuffle(Π[i])
                if length(Π[i]) > 1 && is_dominated(𝒫, Π, i, πi)
                    filter!(πi′ -> πi′ ≠ πi, Π[i])
                    done = false
                    break
                end
            end
        end
    end
end

function is_dominated(𝒫::POMG, Π, i, πi)
    ℐ, 𝒮 = 𝒫.ℐ, 𝒫.𝒮
    jointΠnoti = joint([Π[j] for j in ℐ if j ≠ i])
    π(πi′, πnoti) = [j==i ? πi′ : πnoti[j>i ? j-1 : j] for j in ℐ]
    Ui = Dict((πi′, πnoti, s) => evaluate_plan(𝒫, π(πi′, πnoti), s)[i]
              for πi′ in Π[i], πnoti in jointΠnoti, s in 𝒮)
    model = Model(Ipopt.Optimizer)
    @variable(model, δ)
    @variable(model, b[jointΠnoti, 𝒮] ≥ 0)
    @objective(model, Max, δ)
    @constraint(model, [πi′=Π[i]],
        sum(b[πnoti, s] * (Ui[πi′, πnoti, s] - Ui[πi, πnoti, s])
        for πnoti in jointΠnoti for s in 𝒮) ≥ δ)
    @constraint(model, sum(b) == 1)
    optimize!(model)
    return value(δ) ≥ 0
end
```

Algorithm 26.4. Dynamic programming computes a Nash equilibrium π for a POMG 𝒫, given an initial belief b and horizon depth d. It iteratively computes the policy trees and their expected utilities at each step. The pruning phase at each iteration removes dominated policies, which are policy trees that result in lower expected utility than at least one other available policy tree.

program is feasible, then that means π^i is not dominated by any other $\pi^{i\prime}$:

$$\begin{aligned} \underset{\delta, b}{\text{maximize}} \quad & \delta \\ \text{subject to} \quad & b(\boldsymbol{\pi}^{-i}, s) \geq 0 \text{ for all } \boldsymbol{\pi}^{-i}, s \\ & \sum_{\boldsymbol{\pi}^{-i}} \sum_{s} b(\boldsymbol{\pi}^{-i}, s) = 1 \\ & \sum_{\boldsymbol{\pi}^{-i}} \sum_{s} b(\boldsymbol{\pi}^{-i}, s) \left(U^{\pi^{i\prime}, \boldsymbol{\pi}^{-i}, i}(s) - U^{\pi^{i}, \boldsymbol{\pi}^{-i}, i}(s) \right) \geq \delta \text{ for all } \pi^{i\prime} \end{aligned} \tag{26.6}$$

The pruning step removes dominated policies by randomly selecting an agent i and checking for domination of each of its policies. This process repeats until a pass over all agents fails to find any dominated policies. Example 26.5 shows this process on the multicaregiver crying baby problem.

26.5 Summary

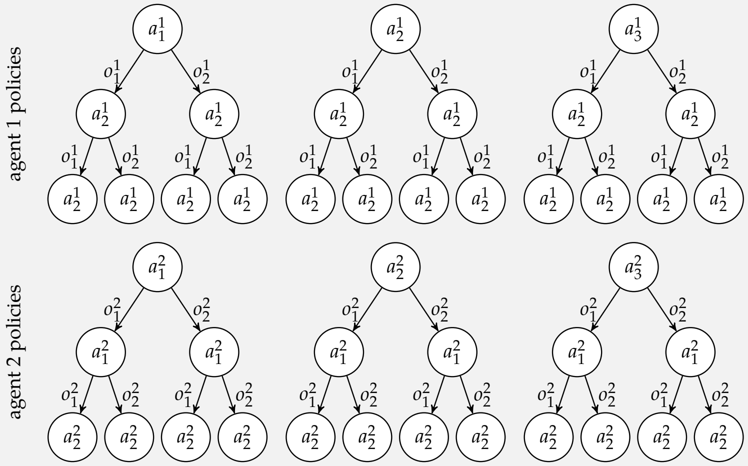
- POMGs generalize POMDPs to multiple agents and MGs to partial observability.
- Because agents generally cannot maintain beliefs in POMGs, policies typically take the form of conditional plans or finite state controllers.
- Nash equilibria, in the form of d-step conditional plans for POMGs, can be obtained by finding Nash equilibria for simple games whose joint actions consist of all possible POMG joint policies.
- Dynamic programming approaches can be used to compute Nash equilibria more efficiently by iteratively constructing sets of deeper conditional plans while pruning dominated plans to restrict the search space.

26.6 Exercises

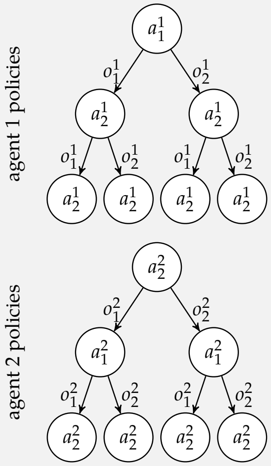
Exercise 26.1. Show that a POMG generalizes both a POMDP and an MG.

Example 26.5. Dynamic programming and a single pruning step for the multicaregiver crying baby problem.

Consider the multicaregiver crying baby problem solved by dynamic programming. Initially, the policies at depth $d = 2$ are



After the pruning step, the agent policies are



In this case, the pruning step finds the best joint policy. This approach significantly reduces the number of possible joint policies that the next iteration of the algorithm needs to consider.

Solution: For any POMDP, we can define a POMG with one agent $\mathcal{I} = \{1\}$. States $\mathcal{S}$ are identical, as are actions $\mathbf{A} = (\mathcal{A}^1)$ and observations $\mathbf{O} = (\mathcal{O}^1)$. Thus, the state transition, observation function, and rewards of the POMG directly follow. The Nash equilibrium optimization has only one agent, so it results in a simple maximization of expected value, which is identical to a POMDP.

For any MG, we can define a POMG with the same agents $\mathcal{I}$, states $\mathcal{S}$, joint actions $\mathbf{A}$, transitions T, and joint rewards $\mathbf{R}$. The individual observations are assigned to be states $\mathcal{O}^i = \mathcal{S}$. The observation function then deterministically provides each agent with the true state $O(\mathbf{o} \mid \mathbf{a}, s') = 1$ if $\mathbf{o} = (s', \ldots, s')$, and 0 otherwise.

Exercise 26.2. How can we incorporate communication between agents into the POMG framework?

Solution: The action space for the agents can be augmented to include communication actions. The other agents can observe these communication actions according to their observation model.

Exercise 26.3. Do agents always have an incentive to communicate?

Solution: Agents in POMGs are often competitive, in which case there would be no incentive to communicate with others. If their rewards are aligned to some degree, they may be inclined to communicate.

Exercise 26.4. How many possible joint conditional plans are there of depth d?

Solution: Recall that there are $|\mathcal{A}|^{(|\mathcal{O}|^d - 1)/(|\mathcal{O}| - 1)}$ possible d-step single-agent conditional plans. We can construct a joint policy of conditional plans using every combination of these single-agent conditional plans across agents. The number of d-step multiagent conditional plans is

$$\prod_{i \in \mathcal{I}} |\mathcal{A}^i|^{(|\mathcal{O}^i|^d - 1)/(|\mathcal{O}^i| - 1)}$$

Exercise 26.5. Define the best response for a POMG in terms of an agent i's utilities $U^{\boldsymbol{\pi},i}$. Propose the iterated best response for POMGs.

Solution: The best response π^i of agent i to other agents' policies $\boldsymbol{\pi}^{-i}$ is defined following equation (24.2) for an initial belief b:

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

with any other policy $\pi^{i\prime}$. For conditional plans, $U^{\boldsymbol{\pi},i}$ is defined by equations (26.1) and (26.2).

The implementation of iterated best response follows from section 24.2.1. First, the conditional plans and simple game can be created, as in algorithm 26.3. Then, we can iterate best response using algorithm 24.8.