27 Collaborative Agents

Many multiagent domains are collaborative, where all agents act independently in an environment while working towards a common shared objective. Applications range from robotic search and rescue to interplanetary exploration rovers. The *decentralized partially observable Markov decision process* (*Dec-POMDP*) captures the generality of POMGs while focusing on such collaborative agent settings.[1] The model is more amenable to scalable approximate algorithms because of its single shared objective, as opposed to a finding an equilibrium among multiple individual agent objectives. This chapter presents the Dec-POMDP model, highlights its subclasses, and describes algorithms that solve them optimally and approximately.

[1] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, ''The Complexity of Decentralized Control of Markov Decision Processes,'' *Mathematics of Operation Research*, vol. 27, no. 4, pp. 819–840, 2002. A more comprehensive introduction is provided by F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs*. Springer, 2016.

27.1 Decentralized Partially Observable Markov Decision Processes

A Dec-POMDP (algorithm 27.1) is a POMG with all agents sharing the same objective. Each agent $i \in \mathcal{I}$ selects an local action $a^i \in \mathcal{A}^i$ and based on a history of local observations $o^i \in \mathcal{O}^i$. The true state of the system $s \in \mathcal{S}$ is shared by all agents. A single reward is generated by $R(s, \mathbf{a})$ based on the state s and the joint action $\mathbf{a}$. The goal of all agents is to maximize the shared expected reward over time under local partial observability. Example 27.1 describes a Dec-POMDP version of the predator-prey problem.

Consider a predator-prey hex world problem in which a team of predators $\mathcal{I}$ strive to capture a single fleeing prey. The predators each move independently. The prey moves randomly to a neighboring cell not occupied by a predator. The predators must work together to capture the prey.

Example 27.1. The collaborative predator-prey problem as a Dec-POMDP. Additional detail is provided in appendix F.15.

Many of the same challenges of POMGs persist in Dec-POMDPs, such as the general inability of agents to maintain a belief state. We focus on policies represented as conditional plans or controllers. My can use the same algorithms introduced in the previous chapter to evaluate policies. All that is required is to create a POMG with $R^i(s, \mathbf{a})$ for each agent i equal to the $R(s, \mathbf{a})$ from the Dec-POMDP.

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

Algorithm 27.1. Data structure for a decentralized partially observable Markov decision process (Dec-POMDP). The `joint` function from algorithm 24.2 allows the creation of all permutations of a set provided, such as `𝒜` or `𝒪`. The `tensorform` function converts the Dec-POMDP `𝒫` to a tensor representation.

27.2 Subclasses

There are many notable subclasses of Dec-POMDPs. Categorizing these subclasses is useful when designing algorithms that take advantage of their specific structure. Table 27.1 summarizes some of these subclasses. Figure 27.1 illustrates the relationships between the models discussed in this book.

Agents	Observability	Communication	Model
Single	Full	—	MDP
Single	Partial	—	POMDP
Multiple	Full	Free	MMDP
Multiple	Full	General	MMDP
Multiple	Joint full	Free	MMDP
Multiple	Joint full	General	Dec-MDP
Multiple	Partial	Free	MPOMDP
Multiple	Partial	General	Dec-POMDP

Table 27.1. Dec-POMDP subclasses categorized by type and computational complexity.

One attribute of interest is *joint full observability*, which is when each agent observes an aspect of the state, such that if they were to combine their observations,

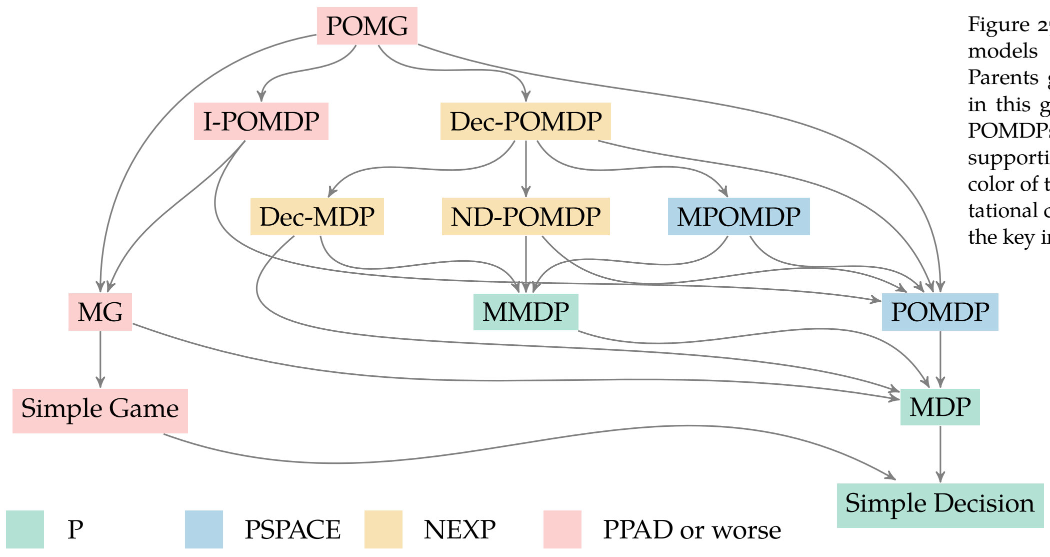


Figure 27.1. A taxonomy for the models discussed in this book. Parents generalize their children in this graph. For example, Dec-POMDPs generalize POMDPs by supporting multiple agents. The color of the nodes indicate computational complexity as indicated in the key in bottom left of the figure.

it would uniquely reveal the true state. The agents, however, do not share their observations. This property ensures that if $O(\mathbf{o} \mid \mathbf{a}, s') > 0$ then $P(s' \mid \mathbf{o}) = 1$. A Dec-POMDP with joint full observability is called a *decentralized Markov decision process* (*Dec-MDP*).

In many settings, the state space of a Dec-POMDP is factored, one for each agent and one for the environment. This is called a *factored Dec-POMDP*. We have $\mathcal{S} = \mathcal{S}^0 \times \mathcal{S}^1 \times \mathcal{S}^k$, where $\mathcal{S}^i$ is the factored state component associated with agent i and $\mathcal{S}^0$ is the factored state component associated with the general environment. For example, in collaborative predator-prey, each agent has their own a state factor for their location and the position of the prey is associated with the environment component of the state space.

In some problems, a factored Dec-POMDP may have one or more of the following properties:

- *Transition independence*, where agents may not affect each other's state:

$$T(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) = T^0(s^{0\prime} \mid s^0) \prod_i T^i(s^{i\prime} \mid s^i, a^i) \qquad (27.1)$$

Independence	Complexity
Transitions, observations, and rewards	P-complete
Transitions and observations	NP-complete
Any other subset	NEXP-complete

Table 27.2. The complexity of factored Dec-POMDPs with different independence assumptions.

- *Observation independence*, where the observations of agents depend only on their local state and actions:

$$O(\mathbf{o} \mid \mathbf{a}, \mathbf{s}') = \prod_i O^i(o^i \mid a^i, s^{i\prime}) \tag{27.2}$$

- *Reward independence*, where the reward can be decomposed into multiple independent pieces that are combined together:[2]

$$R(\mathbf{s}, \mathbf{a}) = R^0(s^0) + \sum_i R^i(s^i, a^i) \tag{27.3}$$

[2] Here, we show the combination of the reward components as a summation, but any monotonically non-decreasing function can be used instead and preserve reward independence.

Depending on which of these independence properties are satisfied, the resulting computational complexity can vary significantly as summarized in table 27.2. It is important to take these independences into account when modeling a problem to improve scalability.

A *network distributed partially observable Markov decision process* (*ND-POMDP*) is a Dec-POMDP with transition and observation independence and a special reward structure. The reward structure is represented by a *coordination graph*. In contrast with the graphs used earlier in this book, a coordination graph is a type of *hypergraph*, which allows edges to connect any number of nodes. The nodes in the ND-POMDP hypergraph correspond to the various agents. The edges relate to interactions between the agents in the reward function. An ND-POMDP associates with each edge j in the hypergraph a reward component R_j that depends on the state and action components to which the edge connects. The reward function in an ND-POMDP is simply the sum of the reward components associated with the edges. Figure 27.2 provides an example coordination graph that results in a reward function that can be decomposed as follows:

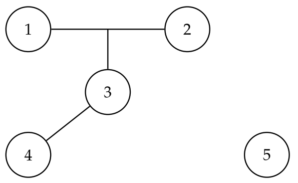


Figure 27.2. An example ND-POMDP structure with five agents. There are three hyper edges: one involving agents 1, 2, and 3; another involving agents 3 and 4; and another involving agent 5 on its own.

$$R_{123}(s_1, s_2, s_3, a_1, a_2, a_3) + R_{34}(s_3, s_4, a_3, a_4) + R_5(s_5, a_5) \tag{27.4}$$

Sensor network and target tracking problems are often framed as ND-POMDPs.

The ND-POMDP model is similar to the transition and observation independent Dec-MDP model, but it does not make the joint full observability assumption. Even if all observations are shared, the true state of the world may not be known. Furthermore, even with factored transitions and observations, a policy in an ND-POMDP is a mapping from observation histories to actions, unlike the transition and observation Dec-MDP case in which policies are mappings from local states to actions. The worst-case complexity remains the same as a full Dec-POMDP (NEXP-complete), but algorithms for ND-POMDPs are typically much more scalable in the number of agents. Scalability can increase as the coordination graph becomes less connected.

If agents are able to communicate their actions and observations perfectly without penalty, the agents are then able to maintain a collective belief state. This model is called a *multiagent MDP* (*MMDP*) or a *multiagent POMDP* (*MPOMDP*). MMDPs and MPOMDPs can also result when when there is transition, observation, and reward independence. Any MDP or POMDP algorithm from earlier chapters can be applied to solve these problems.

27.3 Dynamic Programming

The *dynamic programming* algorithm for Dec-POMDPs applies the Bellman equation at each step and prunes dominated policies.This process is identical to dynamic programming for POMGs, except that each agent shares the same reward. Algorithm 27.2 implements this procedure.

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

function solve(M::DecPOMDPDynamicProgramming, 𝒫::DecPOMDP)
    ℐ, 𝒮, 𝒜, 𝒪, T, O, R, γ = 𝒫.ℐ, 𝒫.𝒮, 𝒫.𝒜, 𝒫.𝒪, 𝒫.T, 𝒫.O, 𝒫.R, 𝒫.γ
    R′(s, a) = [R(s, a) for i in ℐ]
    𝒫′ = POMG(γ, ℐ, 𝒮, 𝒜, 𝒪, T, O, R′)
    M′ = POMGDynamicProgramming(M.b, M.d)
    return solve(M′, 𝒫′)
end
```

Algorithm 27.2. Dynamic programming computes the optimal joint policy π for a Dec-POMDP 𝒫, given an initial belief b and horizon depth d. Since Dec-POMDPs are a special collaborative class of POMGs, we can directly use the POMG algorithm.

27.4 Iterated Best Response

Instead of exploring joint policies directly, we can perform a form of *iterated best response* (algorithm 27.3). In this approach, we iteratively select an agent and compute a best response policy assuming that the other agents are following a fixed policy.[3] This approximate algorithm is typically fast because it is only choosing the best policy for one agent at a time. Moreover, since all agents share the same reward, it tends to terminate after relatively few iterations.

[3] This type of algorithm is also called joint equilibrium-based search for policies (JESP). R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella, "Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings," in *International Joint Conference on Artificial Intelligence* (*IJCAI*), 2003. It can be further improved by performing dynamic programming.

Iterated best response begins with a random initial joint policy $\boldsymbol{\pi}_1$. The process randomly iterates over the agents. If agent i is selected, its policy π^i is updated with a best response to the other agents' fixed policies $\boldsymbol{\pi}^{-i}$ with initial belief distribution b:

$$\pi^i \leftarrow \underset{\pi^{i\prime}}{\arg\max}\, U^{\pi^{i\prime}, \boldsymbol{\pi}^{-i}}(b) \tag{27.5}$$

with ties favoring the current policy. This process can terminate when agents stop changing their policy.

While this algorithm is fast and is guaranteed to converge, it does not always find the best joint policy. It relies on iterated best response to find a Nash equilibrium, but there may be many Nash equilibria with different utilities associated with them. This approach will only find one of them.

27.5 Heuristic Search

Instead of expanding all joint policies, *heuristic search* (algorithm 27.4) explores a fixed number of policies.[4] The fixed number of policies stored over iterations prevents exponential growth. The heuristic exploration guides the search by attempting to only expand the best joint policies until depth d is reached.

[4] This approach is also known as memory bounded dynamic programming (MBDP) S. Seuken and S. Zilberstein, "Memory-Bounded Dynamic Programming for DEC-POMDPs," in *International Joint Conference on Artificial Intelligence* (*IJCAI*), 2007. There are other heuristic search algorithms such as multiagent A* (MMA*). D. Szer, F. Charpillet, and S. Zilberstein, "MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs," in *Conference on Uncertainty in Artificial Intelligence* (*UAI*), 2005.

Each iteration k of the algorithm keeps a set of joint policies $\boldsymbol{\Pi}_k$. Initially this is the one-step conditional plans. Subsequent iterations begin by fully expanding the conditional plans. The goal is to add a fixed number of these for the next iteration.

To decide which among the possible conditional plans to add the set, we want to prioritize the policies that are more likely to maximize utility. However, since we expand the conditional plans from the bottom up, we cannot simply evaluate the policies from the initial belief state b. Instead, we need an estimate of the belief $d - k$ steps into the future, which we compute by taking random actions and

```
struct DecPOMDPIteratedBestResponse
    b     # initial belief
    d     # depth of conditional plans
    k_max # number of iterations
end

function solve(M::DecPOMDPIteratedBestResponse, 𝒫::DecPOMDP)
    ℐ, 𝒮, 𝒜, 𝒪, T, O, R, γ = 𝒫.ℐ, 𝒫.𝒮, 𝒫.𝒜, 𝒫.𝒪, 𝒫.T, 𝒫.O, 𝒫.R, 𝒫.γ
    b, d, k_max = M.b, M.d, M.k_max
    R′(s, a) = [R(s, a) for i in ℐ]
    𝒫′ = POMG(γ, ℐ, 𝒮, 𝒜, 𝒪, T, O, R′)
    Π = create_conditional_plans(𝒫, d)
    π = [rand(Π[i]) for i in ℐ]
    for k in 1:k_max
        for i in shuffle(ℐ)
            π′(πi) = Tuple(j == i ? πi : π[j] for j in ℐ)
            Ui(πi) = utility(𝒫′, b, π′(πi))[i]
            π[i] = _argmax(Ui, Π[i])
        end
    end
    return Tuple(π)
end
```

Algorithm 27.3. Iterated best response for collaborative Dec-POMDP 𝒫 iteratively performs a deterministic best response to rapidly search the conditional plan policies space. The `solve` function implements this procedure up to `k_max` steps, maximizing the value at an initial belief `b` for conditional plans of depth `d`.

simulating state and observations, updating the belief along the way. This belief at iteration k is denoted b_k. For each available joint policy $\boldsymbol{\pi} \in \times_i \Pi_k^i$, the utility $U^{\boldsymbol{\pi}}(b_k)$ is examined to find a utility-maximizing joint policy to add. Example 27.2 demonstrates the process.

27.6 Nonlinear Programming

We can use *nonlinear programming* (*NLP*) (algorithm 27.5) to find an optimal joint controller policy representation of a fixed-size.[5] This method generalizes the NLP approach for POMDPs from section 23.3. Given a fixed set of nodes X^i for each agent i, initial belief b, and initial joint nodes $\mathbf{x}_1$, the optimization problem is:

[5] C. Amato, D. S. Bernstein, and S. Zilberstein, "Optimizing Fixed-Size Stochastic Controllers for POMDPs and Decentralized POMDPs," *Autonomous Agents and Multi-Agent Systems*, vol. 21, no. 3, pp. 293–320, 2010.

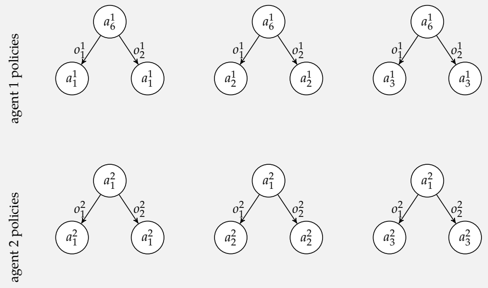
```
struct DecPOMDPHeuristicSearch
    b     # initial belief
    d     # depth of conditional plans
    π_max # number of policies
end

function solve(M::DecPOMDPHeuristicSearch, 𝒫::DecPOMDP)
    ℐ, 𝒮, 𝒜, 𝒪, T, O, R, γ = 𝒫.ℐ, 𝒫.𝒮, 𝒫.𝒜, 𝒫.𝒪, 𝒫.T, 𝒫.O, 𝒫.R, 𝒫.γ
    b, d, π_max = M.b, M.d, M.π_max
    R′(s, a) = [R(s, a) for i in ℐ]
    𝒫′ = POMG(γ, ℐ, 𝒮, 𝒜, 𝒪, T, O, R′)
    Π = [[ConditionalPlan(ai) for ai in 𝒜[i]] for i in ℐ]
    for t in 1:d
        allΠ = expand_conditional_plans(𝒫, Π)
        Π = [[] for i in ℐ]
        for z in 1:π_max
            b′ = explore(M, 𝒫, t)
            π = _argmax(π -> first(utility(𝒫′, b′, π)), joint(allΠ))
            for i in ℐ
                push!(Π[i], π[i])
                filter!(πi -> πi != π[i], allΠ[i])
            end
        end
    end
    return _argmax(π -> first(utility(𝒫′, b, π)), joint(Π))
end

function explore(M::DecPOMDPHeuristicSearch, 𝒫::DecPOMDP, t)
    ℐ, 𝒮, 𝒜, 𝒪, T, O, R, γ = 𝒫.ℐ, 𝒫.𝒮, 𝒫.𝒜, 𝒫.𝒪, 𝒫.T, 𝒫.O, 𝒫.R, 𝒫.γ
    b = copy(M.b)
    b′ = similar(b)
    s = rand(SetCategorical(𝒮, b))
    for τ in 1:t
        a = Tuple(rand(𝒜i) for 𝒜i in 𝒜)
        s′ = rand(SetCategorical(𝒮, [T(s,a,s′) for s′ in 𝒮]))
        o = rand(SetCategorical(joint(𝒪), [O(a,s′,o) for o in joint(𝒪)]))
        for (i′, s′) in enumerate(𝒮)
            po = O(a, s′, o)
            b′[i′] = po*sum(T(s,a,s′)*b[i] for (i,s) in enumerate(𝒮))
        end
        normalize!(b′, 1)
        b, s = b′, s′
    end
    return b′
end
```

Algorithm 27.4. Memory bounded heuristic search uses a heuristic function to search the space of conditional plans for a Dec-POMDP 𝒫. The `solve` function tries to maximize the value at an initial belief `b` for joint conditional plans of depth `d`. The `explore` function generates a belief `t` steps into the future by taking random actions and simulating actions and observations. The algorithm is memory-bounded, only keeping `π_max` conditional plans per agent.

Consider the collaborative predator-prey problem shown in the margin. We apply heuristic search to a depth of $d = 3$ with 3 policies retained at each iteration. After iteration $k = 1$, the policies are:



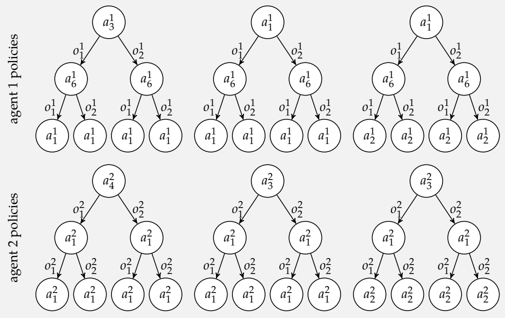
At the next iteration $k = 2$, heuristic search again starts at the initial belief and takes $d - k = 3 - 2 = 1$ steps following the heuristic exploration. The explored beliefs used to select the next three conditional plans are:

$$b_3 = [0.0, 0.03, 0.01, 0.0, 0.03, 0.01, 0.0, 0.15, 0.05 \\ 0.0, 0.01, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.03 \\ 0.06, 0.11, 0.32, 0.06, 0.03, 0.01, 0.01, 0.04, 0.06]$$

$$b_2 = [0.0, 0.21, 0.03, 0.0, 0.04, 0.01, 0.0, 0.05, 0.01 \\ 0.0, 0.08, 0.03, 0.0, 0.0, 0.01, 0.0, 0.0, 0.01 \\ 0.08, 0.34, 0.03, 0.02, 0.05, 0.01, 0.0, 0.01, 0.0]$$

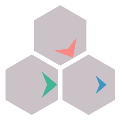
$$b_1 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.17, 0.0, 0.03 \\ 0.01, 0.0, 0.0, 0.05, 0.0, 0.01, 0.23, 0.0, 0.08 \\ 0.01, 0.0, 0.0, 0.14, 0.0, 0.03, 0.22, 0.0, 0.01]$$

The policies after iteration $k = 2$ are:



The beliefs were used to determine the root node's action and the two subtrees below it. These subtrees are built from the prior iteration's trees.

Example 27.2. Heuristic search exploration and conditional plan expansion for the collaborative predator-prey hex world problem shown below. The predators are red and green. The prey is blue..



$$
\begin{aligned}
\underset{U,\boldsymbol{\psi},\boldsymbol{\eta}}{\text{maximize}} \quad & \sum_s b(s) U(\mathbf{x}_1, s) \\
\text{subject to} \quad & U(\mathbf{x}, s) = \sum_{\mathbf{a}} \prod_i \psi^i(a^i \mid x^i) \left(R(s, \mathbf{a}) + \gamma \sum_{s'} T(s' \mid s, \mathbf{a}) \sum_{\mathbf{o}} O(\mathbf{o} \mid \mathbf{a}, s') \sum_{\mathbf{x}'} \prod_i \eta^i(x^{i\prime} \mid x^i, a^i, o^i) U(\mathbf{x}', s') \right) \\
& \qquad \text{for all } \mathbf{x}, s \\
& \psi^i(a^i \mid x^i) \geq 0 \quad \text{for all } i, x^i, a^i \\
& \sum_a \psi^i(a^i \mid x^i) = 1 \quad \text{for all } i, x^i \\
& \eta^i(x^{i\prime} \mid x^i, a^i, o^i) \geq 0 \quad \text{for all } i, x^i, a^i, o^i, x^{i\prime} \\
& \sum_{x^{i\prime}} \eta^i(x^{i\prime} \mid x^i, a^i, o^i) = 1 \quad \text{for all } i, x^i, a' o^i
\end{aligned}
\tag{27.6}
$$

27.7 *Summary*

- Decentralized partially observable Markov decision processes (Dec-POMDPs) are fully cooperative POMGs that model a team of agents working together towards a shared goal, each acting individually using only local information.

- Because determining a belief state is infeasible as in POMGs, policies are generally represented as conditional plans or controllers, allowing each agent to map individual sequences of observations to individual actions.

- Many subclasses of Dec-POMDPs exist with different degrees of computational complexity.

- Dynamic programming computes the value function iteratively, pruning dominated policies as it iterates using a linear program.

- Iterated best response computes a best utility-maximizing response policy for a single agent at a time, iteratively converging to a joint equilibrium.

- Heuristic search searches a fixed subset of policies at each iteration, guided by a heuristic.

- Nonlinear programming can be used to generate controllers of a fixed size.

```
struct DecPOMDPNonlinearProgramming
    b # initial belief
    ℓ # number of nodes for each agent
end

function tensorform(𝒫::DecPOMDP)
    ℐ, 𝒮, 𝒜, 𝒪, R, T, O = 𝒫.ℐ, 𝒫.𝒮, 𝒫.𝒜, 𝒫.𝒪, 𝒫.R, 𝒫.T, 𝒫.O
    ℐ′ = eachindex(ℐ)
    𝒮′ = eachindex(𝒮)
    𝒜′ = [eachindex(𝒜i) for 𝒜i in 𝒜]
    𝒪′ = [eachindex(𝒪i) for 𝒪i in 𝒪]
    R′ = [R(s,a) for s in 𝒮, a in joint(𝒜)]
    T′ = [T(s,a,s′) for s in 𝒮, a in joint(𝒜), s′ in 𝒮]
    O′ = [O(a,s′,o) for a in joint(𝒜), s′ in 𝒮, o in joint(𝒪)]
    return ℐ′, 𝒮′, 𝒜′, 𝒪′, R′, T′, O′
end

function solve(M::DecPOMDPNonlinearProgramming, 𝒫::DecPOMDP)
    𝒫, γ, b = 𝒫, 𝒫.γ, M.b
    ℐ, 𝒮, 𝒜, 𝒪, R, T, O = tensorform(𝒫)
    X = [collect(1:M.ℓ) for i in ℐ]
    jointX, joint𝒜, joint𝒪 = joint(X), joint(𝒜), joint(𝒪)
    x1 = jointX[1]
    model = Model(Ipopt.Optimizer)
    @variable(model, U[jointX,𝒮])
    @variable(model, ψ[i=ℐ,X[i],𝒜[i]] ≥ 0)
    @variable(model, η[i=ℐ,X[i],𝒜[i],𝒪[i],X[i]] ≥ 0)
    @objective(model, Max, b⋅U[x1,:])
    @NLconstraint(model, [x=jointX,s=𝒮],
        U[x,s] == (sum(prod(ψ[i,x[i],a[i]] for i in ℐ)
                  *(R[s,y] + γ*sum(T[s,y,s′]*sum(O[y,s′,z]
                      *sum(prod(η[i,x[i],a[i],o[i],x′[i]] for i in ℐ)
                              *U[x′,s′] for x′ in jointX)
                      for (z, o) in enumerate(joint𝒪)) for s′ in 𝒮))
                  for (y, a) in enumerate(joint𝒜))))
    @constraint(model, [i=ℐ,xi=X[i]],
                sum(ψ[i,xi,ai] for ai in 𝒜[i]) == 1)
    @constraint(model, [i=ℐ,xi=X[i],ai=𝒜[i],oi=𝒪[i]],
                sum(η[i,xi,ai,oi,xi′] for xi′ in X[i]) == 1)
    optimize!(model)
    ψ′, η′ = value.(ψ), value.(η)
    return [ControllerPolicy(𝒫, X[i],
            Dict((xi,𝒫.𝒜[i][ai]) => ψ′[i,xi,ai]
                 for xi in X[i], ai in 𝒜[i]),
            Dict((xi,𝒫.𝒜[i][ai],𝒫.𝒪[i][oi],xi′) => η′[i,xi,ai,oi,xi′]
                 for xi in X[i], ai in 𝒜[i], oi in 𝒪[i], xi′ in X[i]))
        for i in ℐ]
end
```

Algorithm 27.5. Nonlinear programming (NLP) computes the optimal joint controller policy π for a Dec-POMDP 𝒫, given an initial belief b and number of controller nodes ℓ for each agent. This generalizes the NLP solution in algorithm 23.5.

27.8 Exercises

Exercise 27.1. Compared to a POSG, why might a Dec-POMDP be more useful in practice, especially in robotic applications?

Solution: Dec-POMDPs are fully-cooperative. Multi-robot domains, such as search and rescue, require the robots to work together as a team. The engineers have full control of the executable policies of each robot. Conversely, in a competitive POMG (or even MG), the algorithms compute policies for all agents. However, in competitive settings, we likely only have control over one agent's policy. Additionally, the model realistically captures the limited sensors and sensor models in robotics, as in POMDPs, except in the multiagent setting.

Exercise 27.2. Why is a Dec-MDP with joint full observability different from agents knowing the state?

Solution: Full joint observability means if agents were to share their individual observations, then the team would know the true state. This can be done offline during planning. Thus in Dec-MDPs, the true state is essentially known during planning. The issue is that it requires agents to share their individual observations, which cannot be done online during execution. Therefore, planning still needs to reason about the uncertain observations made by the other agents.

Exercise 27.3. Propose a fast algorithm for a Dec-MDP with transition, observation, and reward independence. Prove that it is correct.

Solution: If a factored Dec-MDP satisfies all three independence assumptions, then we can solve it as $|\mathcal{I}|$ separate MDPs. The resulting policy π^i for each agent i's MDP can then be combined to form the optimal joint policy. To prove this fact, consider the utility of each agent's individual MDP:

$$U^{\pi^i}(s^i) = R\left(s^i, \pi^i()\right) + \gamma \left[\sum_{s^{i\prime}} T^i\left(s^{i\prime} \mid s^i, \pi^i()\right) \sum_{o^i} O^i\left(o^i \mid \pi^i(), s^{i\prime}\right) U^{\pi^i(o^i)}(s^{i\prime}) \right] \tag{27.7}$$

We sum of each of their individual contributions:

$$\sum_i U^{\pi^i}(s) = \sum_i \left[R\left(s^i, \pi^i()\right) + \gamma \left[\sum_{s^{i\prime}} T^i\left(s^{i\prime} \mid s^i, \pi^i()\right) \sum_{o^i} O^i\left(o^i \mid \pi^i(), s^{i\prime}\right) U^{\pi^i(o^i)}(s^{i\prime}) \right] \right] \tag{27.8}$$

We can combine T^i and O^i into a single probability distribution P, move the summation, and apply the definition of reward independence:

$$\sum_i U^{\pi^i}(s) = \sum_i \left[R\left(s^i, \pi^i()\right) + \gamma \left[\sum_{s^{i\prime}} P\left(s^{i\prime} \mid s^i, \pi^i()\right) \sum_{o^i} P\left(o^i \mid \pi^i(), s^{i\prime}\right) U^{\pi^i(o^i)}(s^{i\prime}) \right] \right] \tag{27.9}$$

$$= \sum_i R\left(s^i, \pi^i()\right) + \sum_i \left[\gamma \left[\sum_{s^{i\prime}} P\left(s^{i\prime} \mid s^i, \pi^i()\right) \sum_{o^i} P\left(o^i \mid \pi^i(), s^{i\prime}\right) U^{\pi^i(o^i)}(s^{i\prime}) \right] \right] \tag{27.10}$$

$$= R(s, \pi()) + \sum_i \left[\gamma \left[\sum_{s^{i\prime}} P\left(s^{i\prime} \mid s^i, \pi^i()\right) \sum_{o^i} P\left(o^i \mid \pi^i(), s^{i\prime}\right) U^{\pi^i(o^i)}(s^{i\prime}) \right] \right] \tag{27.11}$$

Now, we marginalize over all successors s and observations o. Because of the transition and observation independence, we can freely condition the distributions on these other non-i state and observation factors, which is the same as conditioning on s and o. We can then apply the definition of transition and observation independence. Lastly, we can move the summation in and recognize $U^{\boldsymbol{\pi}}(s)$ results:

$$\sum_i U^{\pi^i}(s) = R(s, \boldsymbol{\pi}()) + \sum_i \left[\gamma \left[\sum_{s'} P\left(s' \mid s^i, \pi^i()\right) \sum_{o} P\left(o \mid \pi^i(), s^{i\prime}\right) U^{\pi^i(o^i)}(s^{i\prime}) \right] \right] \tag{27.12}$$

$$= R(s, \boldsymbol{\pi}()) + \sum_i \left[\gamma \left[\sum_{s'} P\left(s^{0\prime} \mid s^0\right) \prod_j P\left(s^{j\prime} \mid s^i, \pi^i()\right) \sum_o \prod_j P\left(o^j \mid \pi^i(), s^{i\prime}\right) U^{\pi^i(o^i)}(s^{i\prime}) \right] \right] \tag{27.13}$$

$$= R(s, \boldsymbol{\pi}()) + \sum_i \left[\gamma \left[\sum_{s'} P\left(s^{0\prime} \mid s^0\right) \prod_j P\left(s^{j\prime} \mid s, \pi()\right) \sum_o \prod_j P\left(o^j \mid \pi(), s'\right) U^{\pi^i(o^i)}(s^{i\prime}) \right] \right] \tag{27.14}$$

$$= R(s, \boldsymbol{\pi}()) + \sum_i \left[\gamma \left[\sum_{s'} T(s' \mid s, \boldsymbol{\pi}()) \sum_o O(o \mid \boldsymbol{\pi}(), s') U^{\pi^i(o^i)}(s^{i\prime}) \right] \right] \tag{27.15}$$

$$= R(s, \boldsymbol{\pi}()) + \gamma \left[\sum_{s'} T(s' \mid s, \boldsymbol{\pi}()) \sum_o O(o \mid \boldsymbol{\pi}(), s') \left[\sum_i U^{\pi^i(o^i)}(s^{i\prime}) \right] \right] \tag{27.16}$$

$$= R(s, \boldsymbol{\pi}()) + \gamma \left[\sum_{s'} T(s' \mid s, \boldsymbol{\pi}()) \sum_o O(o \mid \boldsymbol{\pi}(), s') U^{\boldsymbol{\pi}(o)}(s') \right] \tag{27.17}$$

$$= U^{\boldsymbol{\pi}}(s) \tag{27.18}$$

This is the Dec-MDP utility function derived from equation (26.1), completing the proof.

Exercise 27.4. How can we use an MMDP or MPOMDP as a heuristic in Dec-POMDP heuristic search?

Solution: We can assume free communication for planning. At each time step t, all agents know $\mathbf{a}_t$ and $\mathbf{o}_t$, allowing us to maintain a multiagent belief b_t, resulting in an MPOMDP. This MPOMDP solution can be used as a heuristic to guide the search of policy trees. Alternatively, we create a heuristic where we assume that the true state and joint actions known. This results in an MMDP, and can also be used as a heuristic. These assumptions are only used for planning. Execution is still a Dec-POMDP wherein agents receive individual observations without free communication among themselves. Either heuristic results in a joint policy $\hat{\boldsymbol{\pi}}$ for heuristic exploration.

Exercise 27.5. How can we compute a best response controller? Describe how this could be used in iterated best response.

Solution: For an agent i, the best response controller X^i, ψ^i, and η^i can be computed by a nonlinear program. The program is similar to section 27.6, except that $\mathbf{X}^{-i}$, $\boldsymbol{\psi}^{-i}$, and $\boldsymbol{\eta}^{-i}$ are now given and no longer variables:

$$\begin{aligned}
\underset{U,\psi^i,\eta^i}{\text{maximize}} \quad & \sum_s b(s)U(\mathbf{x}_1, s) \\
\text{subject to} \quad & U(\mathbf{x}, s) = \sum_{\mathbf{a}} \prod_i \psi^i(a^i \mid x^i) \left(R(s, \mathbf{a}) + \gamma \sum_{s'} T(s' \mid s, \mathbf{a}) \sum_{\mathbf{o}} O(\mathbf{o} \mid \mathbf{a}, s') \sum_{\mathbf{x}'} \prod_i \eta^i(x^{i\prime} \mid x^i, a^i, o^i) U(\mathbf{x}', s') \right) \\
& \qquad \text{for all } \mathbf{x}, s \\
& \psi^i(a^i \mid x^i) \geq 0 \quad \text{for all } x^i, a^i \\
& \sum_a \psi^i(a^i \mid x^i) = 1 \quad \text{for all } x^i \\
& \eta^i(x^{i\prime} \mid x^i, a^i, o^i) \geq 0 \quad \text{for all } x^i, a^i, o^i, x^{i\prime} \\
& \sum_{x^{i\prime}} \eta^i(x^{i\prime} \mid x^i, a^i, o^i) = 1 \quad \text{for all } x^i, a^\prime o^i
\end{aligned} \tag{27.19}$$

Adapting algorithm 27.3 for controller policies, this program replaces the inner best response operation.

Appendices