7

Cooperative Decision Making

Christopher Amato

Solving problems in which a group of agents must operate collaboratively in sequential environments is an important challenge. As the construction of agents (e.g., robots, sensors, software agents) becomes less costly, more agents can be deployed, but for a team to achieve its full potential, each agent must reason about the others. In this chapter, we discuss models in which agents may have uncertainty about both the state of the environment and the choices of the other agents. Agents seek to optimize a shared objective function, but must develop plans of action that are based on a partial view of the environment. We describe modeling this problem as a decentralized partially observable Markov decision process (Dec-POMDP) and discuss the complexity and salient properties of this model. We also provide an overview of exact and approximate solution methods for Dec-POMDPs, discuss the use of communication between agents in this model, and describe notable subclasses with additional modeling assumptions and reduced complexity.

7.1 Formulation

Multiagent systems can be modeled in a centralized way using MDPs and POMDPs by requiring all agent information and decision making to be centralized at each step. However, many problems require decentralized execution. The Dec-POMDP model is an extension of the MDP and POMDP models that provides decentralized policies for each agent. In a Dec-POMDP, the dynamics of the system and the objective function depend on the actions of all agents, but each agent must make decisions based on local information. We first present the model, discuss an example problem, and outline two forms of solution representations.

7.1.1 Decentralized POMDPs

A Dec-POMDP is defined by the following:

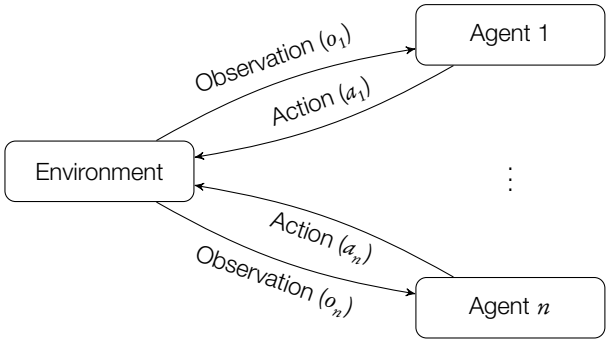


Figure 7.1 Dec-POMDP structure with n agents.

- I, a finite set of agents,
- S, a finite set of states with designated initial state distribution b_0,
- A_i, a finite set of actions for each agent i,
- T, transition probability function $T(s' \mid s, \mathbf{a})$ that specifies the probability of transitioning from state s to s' when action $\mathbf{a}$ is taken by the agents,
- R, a reward function $R(s, \mathbf{a})$ that specifies the immediate reward for being in state s and taking the actions $\mathbf{a}$,
- Ω_i, a finite set of observations for each agent i, and
- O, observation model $O(\mathbf{o} \mid s', \mathbf{a})$ that specifies the probability of observing $\mathbf{o}$ in state s' given action $\mathbf{a}$.

As shown in Figure 7.1, a Dec-POMDP involves multiple agents that operate under uncertainty on the basis of different streams of observations. Like an MDP or a POMDP, a Dec-POMDP unfolds over a finite or infinite sequence of steps. At each step, every agent chooses an action based purely on its local observations, resulting in an immediate reward for the set of agents and an observation for each individual agent. Because the state is not directly observed, it may be beneficial for each agent to remember its observation history. Unlike in POMDPs, in Dec-POMDPs it is not always possible to calculate an estimate of the system state (a belief state) from the observation history of a single agent (as we will discuss in Section 7.2.1).

A *joint policy* is a set of policies, one for each agent in the problem. A *local policy* for an agent is a mapping from local observation histories to actions. The objective in a Dec-POMDP is to find a joint policy that maximizes expected utility. As with MDPs and POMDPs, we can define utility in different ways, such as the sum of rewards over a finite horizon or the discounted sum of rewards over an infinite horizon (Section 4.1.2).

A generalization of the Dec-POMDP formulation involves specifying independent reward functions for the various agents. If the agents are to maximize their own accumulation of reward, then the problem becomes a partially observable stochastic game

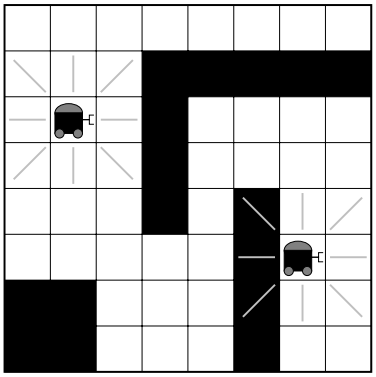


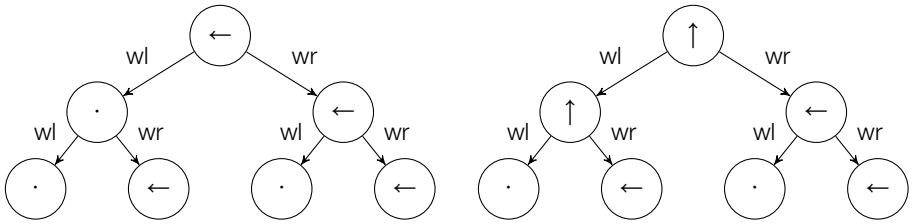
Figure 7.2 Robot navigation problem with two agents.

(POSG). Such problems require a game-theoretic treatment (similar to what was introduced in Section 3.3 for single-shot decisions) and are significantly more difficult to analyze.

7.1.2 Example Problem

One set of domains that can be modeled as Dec-POMDPs is robot navigation and exploration problems. A simple grid-based robot navigation problem is shown in Figure 7.2. The states in this problem correspond to the positions of both robots. The actions are up, down, left, right, and stay in place. The movement actions move the agent one grid cell in the desired direction with probability 0.6 or in one of the other directions or current cell with probability 0.1 each. Movements into a wall result in the robot's staying in place. Choosing to stay in place always keeps the agent in the current location. We assume perfect observation of the grid cells immediately surrounding the agent (as indicated by the gray lines in the figure). As a result, each robot can observe the wall configurations in surrounding squares, but not its own actual location. The objective is for the agents to meet as quickly as possible. When both agents occupy the same square, a reward of one is received; otherwise, no reward is received. The initial state is the one shown in the figure.

There are three types of uncertainty in this problem: uncertainty about action outcomes (as in an MDP), uncertainty about sensor information (as in a POMDP), and uncertainty about the information of the other agents. Although the agents observe the surrounding grid squares and can narrow down their own possible locations, the observations usually provide no information about the choices or location of the other agent. As a result, optimal algorithms will often consider all the possible choices and locations for the other agents when generating a solution. As discussed later, centralized



(a) Agent 1. (b) Agent 2.

Figure 7.3 Policy tree representation for a two-agent Dec-POMDP.

belief states as used by a POMDP are no longer possible, and solving a Dec-POMDP becomes much more difficult. A solution to this problem would be for each agent to move toward a central location and, if the other agent is not seen, continue to some locations the other agent could be. If the other agent is still not seen, the agent could move to a location that was agreed upon in the solution process and wait for the other agent to arrive. The solution method would produce the policy of which movements an agent should make after seeing different observation histories, optimizing these choices over the uncertainty.

7.1.3 Solution Representations

For finite-horizon problems, local policies can be represented by policy trees. An example is shown in Figure 7.3. Such trees are similar to the policy trees for POMDPs, but now each agent possesses its own tree that is independent of the other agents' trees. To make this more concrete, we consider a simplified version of the example problem above in a 2×2 grid without obstacles. Agent 1 starts in the top right, and agent 2 starts in the bottom left. Actions are represented by arrows or the stop symbol, $\cdot$, (each agent can move in the given direction or stay where it is). Observations are labeled "wl" and "wr" for seeing a wall on the left or the right, respectively. In this representation, an agent takes the action defined at the root node and then, after seeing an observation, chooses the next action that is defined by the respective branch. This sequence continues until the action at a leaf node is executed. For example, agent 1 would first move left, and then if a wall is seen on the right, the agent would move left again. If a wall is now seen on the left, the agent does not move on the final step. A policy tree is a record of the entire local history for an agent up to some fixed horizon. Because each tree is independent of the others, it can be executed in a decentralized manner. The resulting policies allow the agents to meet in the top left square quickly and with high probability.

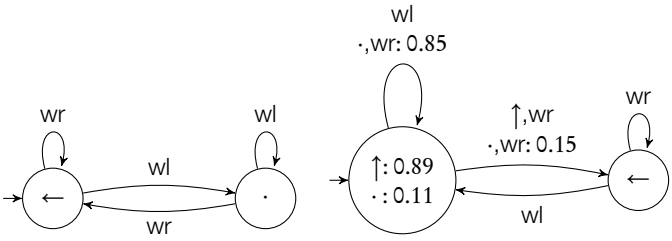


Figure 7.4 Stochastic controller representation for a two-agent Dec-POMDP.

These trees can be evaluated by summing the rewards at each step weighted by the likelihood of transitioning to a given state and observing a given set of observations. For a set of agents, the value of trees $\mathbf{q}$ while starting at state s is given recursively by

$$U(\mathbf{q}, s) = R(\mathbf{a_q}, s) + \sum_{s', \mathbf{o}} P(s' \mid \mathbf{a_q}, s) P(\mathbf{o} \mid \mathbf{a_q}, s') U(\mathbf{q_o}, s'), \tag{7.1}$$

where $\mathbf{a_q}$ are the actions defined at the root of trees $\mathbf{q}$, while $\mathbf{q_o}$ and are the subtrees of $\mathbf{q}$ that are visited after $\mathbf{o}$ have been seen.

While this representation is useful for finite-horizon problems, infinite-horizon problems would require trees of infinite height. Another option is to condition action selection on some internal memory state. These solutions can be represented as a set of local finite-state controllers (seen in Figure 7.4). Again, these controllers are similar to those used by POMDPs except each agent possesses its own independent controller.

The controllers operate in a very similar way to the policy trees in that there is a designated initial node and following the action selection at that node, the controller transitions to the next node depending on the observation seen. This continues for the infinite steps of the problem. We can also consider stochastic controllers, which choose actions and transitions stochastically, as they are able to produce higher-quality solutions than deterministic controllers with the same number of nodes. Throughout this chapter, controller states will be referred to as nodes to help distinguish them from system states.

An example of two-node stochastic controllers for the 2×2 version of the example problem can be seen Figure 7.4. Agent 2 begins at node 1, moving up with probability 0.89 and staying in place with probability 0.11. If the agent stayed in place and a wall is then seen on the left (observation "wl"), on the next step, the controller would transition to node 1 and the agent would use the same distribution of actions again. If a wall was seen on the right instead (observation "wr"), there is a 0.85 probability that the

controller will transition back to node 1 and a 0.15 probability that the controller will transition to node 2 for the next step. The resulting policy again allows the agents to meet quickly and with high probability in the top left square. The finite-state controller allows an infinite-horizon policy to be represented compactly by remembering some aspects of the agent's history without representing the entire local history.

We can evaluate the joint policy by beginning at the initial node and transitioning through the controller according to the actions taken and observations seen. We define the probability an action a_i will be taken in node q_i by agent i to be $P(a_i \mid q_i)$. We also have $P(q_i' \mid q_i, a_i, o_i)$ represent the probability that the controller transitions to node q_i' given that the controller is currently in q_i, takes action a_i, and observes o_i. The value for starting in nodes $\mathbf{q}$ and at state s with action selection and node transition probabilities for each agent, i, is given by the following Bellman equation:

$$U(\mathbf{q}, s) = \sum_{\mathbf{a}} \left(\prod_i P(a_i \mid q_i) \left[R(s, \mathbf{a}) + \gamma \sum_{s', \mathbf{o}, \mathbf{q}'} P(s' \mid \mathbf{a}, s) O(\mathbf{o} \mid s', \mathbf{a}) \prod_j P(q_j' \mid q_j, a_j, o_j) U(\mathbf{q}', s') \right] \right). \quad (7.2)$$

Note that the values (for either the trees or controllers) can be calculated offline in order to determine a policy for each agent that can then be executed online in a decentralized manner. In fact, as we will discuss below, many algorithms consider this offline planning scenario in which the solution (trees or controllers) is generated offline in a centralized manner and then the policy is executed online in a decentralized manner.

7.2 Properties

The decentralized nature of Dec-POMDPs makes them fundamentally different from POMDPs. We explain some of these differences, discuss the complexity of the general Dec-POMDP model, and describe an extension of the concept of belief states to multiagent problems.

7.2.1 Differences with POMDPs

In a Dec-POMDP, the decisions of each agent affect all the agents in the domain, but because of the decentralized nature of the model, each agent must choose actions based solely on local information. Because each agent receives a separate observation that does not usually provide sufficient information to efficiently reason about the other agents, solving a Dec-POMDP optimally becomes very difficult. Each agent may receive a different piece of information that does not allow a common state estimate or any estimate of the other agents' decisions to be calculated. For example, in the

robot navigation example problem in Figure 7.2, even though each agent knows the initial location of the other, agent 1's observations (until it becomes adjacent) give it no information about agent 2's choice of action or location. Therefore, while it may be possible to limit the possible locations the other agent may be in (such as those not in surrounding grid cells), it is usually not possible to generate an estimate of the system state. Exceptions to this are when observations provide this information (e.g., when the other agent is seen) or the policies of the other agents are known (as discussed later).

State estimates are crucial in single-agent problems because they allow the agent's history to be summarized concisely (as belief states), but they are not generally available in Dec-POMDPs. The lack of state estimates (and thus the lack of a concise sufficient statistic) requires agents to remember whole action and observation histories in order to act optimally; therefore, Dec-POMDPs cannot be transformed into belief state MDPs and we must use a different set of tools to solve them.

7.2.2 Dec-POMDP Complexity

The difference between Dec-POMDPs and POMDPs is seen in the complexity of the finite-horizon problem. A Dec-POMDP with at least two agents is *NEXP-complete*, a category of problem that may require doubly exponential time in practice. This complexity is in contrast to the MDP (P-complete) and the POMDP (PSPACE-complete). As in solving an infinite-horizon POMDP, optimally solving an infinite-horizon Dec-POMDP is *undecidable* as it may require infinite resources (infinitely sized controllers), but ϵ-optimal solutions can be found with finite time and memory. These complexity differences show that introducing multiple decentralized agents causes Dec-POMDPs to be significantly more difficult to solve than POMDPs. The intuition behind this complexity is that agents must consider the possible choices of all other agents in addition to the state and action uncertainty present in order to produce an optimal policy.

7.2.3 Generalized Belief States

As mentioned above, from an agent's perspective, not only is there uncertainty about the state, but there may also be uncertainty about the policies of the other agents. If we consider the possible policies of the other agents as part of the state of the system, we can form a *generalized belief state* (sometimes called a multiagent belief state). Agents can also consider the generalized belief space that includes all possible distributions over states of the system and policies of the other agents. In a two-agent situation, the value of an agent's policy p at a given generalized belief state b_G is given by

$$U(p, b_G) = \sum_{q,s} b_G(s, q) U(p, q, s), \tag{7.3}$$

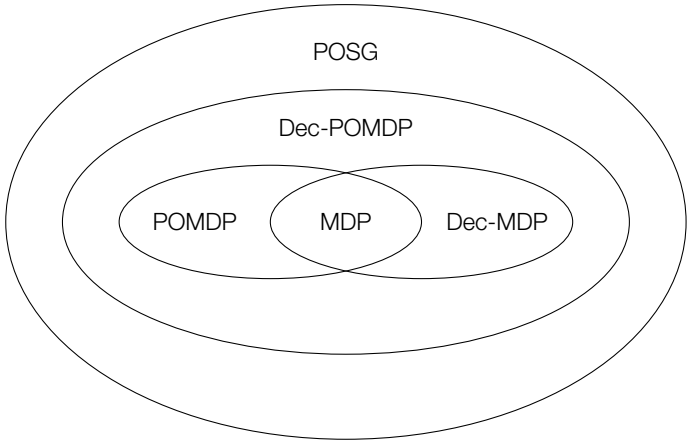


Figure 7.5 Subclasses and superclasses of Dec-POMDPs.

where q represents a policy of the other agent. If all other agents have known policies, the generalized belief state is the same as a POMDP belief state, and we can solve the Dec-POMDP for the remaining agent as a POMDP (the other agents can be thought of as part of the environment). Unfortunately, when we solve a Dec-POMDP, probability distributions for other agent policies are generally not known, and, as a result, the generalized belief state cannot usually be calculated. Nevertheless, the idea of the possible policies of the other agents and the generalized belief space can be used in the decision-making process.

7.3 Notable Subclasses

Because of the high worst-case complexity of Dec-POMDPs, a number of Dec-POMDP subclasses have been explored that may be more tractable theoretically or in practice. This section discusses a few of these, including the Dec-MDP, network distributed POMDP (ND-POMDP), and multiagent MDP (MMDP). The relationships between Dec-POMDPs, POSGs, POMDPs, MDPs, and Dec-MDPs is shown in Figure 7.5.

7.3.1 Dec-MDPs

A *decentralized Markov decision process* (Dec-MDP) is a Dec-POMDP with joint full observability. In other words, if the observations of all the agents are combined, the state of the environment is known exactly. A common example of a Dec-MDP is a problem in which the state consists of the locations of a set of robots and each agent observes its own location perfectly. Therefore, if all these observations are combined, the locations of all robots would be known. Note that (perhaps counterintuitively)

Dec-MDPs do include observations for each agent that are often noisy indicators of the state. The complexity of Dec-MDPs is the same as Dec-POMDPs; although the true state may be known if observations are shared, this sharing does not take place.

We now discuss factorization in the context of Dec-MDPs, but similar factorization can be done in full Dec-POMDPs. A *factored n-agent Dec-MDP* is a Dec-MDP in which the world state can be factored into $n+1$ components, $S = S_0 \times S_1 \times \ldots \times S_n$. The states in S_i are the *local states* associated with agent i. The S_0 component is a property of the environment and is not affected by any agent actions (and is sometimes omitted). For example, S_0 may be the location of a target in a target-tracking scenario. Similarly, an agent's local state, S_i might consist of its location in a grid. A factored, n-agent Dec-MDP is said to be *locally fully observable* if each agent fully observes its own state component.

A factored, n-agent Dec-MDP is said to be *transition independent* if the state transition probabilities factorize as follows:

$$T(s' \mid s, \mathbf{a}) = T_0(s'_0 \mid s_0) \prod_i T_i(s'_i \mid s_i, a_i). \tag{7.4}$$

Here, $T_i(s'_i \mid s_i, a_i)$ represents the probability that the local state of agent i transitions from s_i to s'_i after executing action a_i. The unaffected state transition probability is denoted $T_0(s'_0 \mid s_0)$. The robot navigation problem is transition independent if the robots never affect each other (i.e., they do not bump into each other when moving and can share the same grid cell).

A factored, n-agent Dec-MDP is said to be *observation independent* if the observation probabilities factorize as follows:

$$O(\mathbf{o} \mid s, \mathbf{a}) = \prod_i O_i(o_i \mid s_i, a_i). \tag{7.5}$$

In the equation above, $O_i(o_i \mid s_i, a_i)$ represents the probability that agent i receives observation o_i in state s_i after executing action a_i. If the robots in the navigation problem cannot observe each other (due to working in different locations or lack of sensors), the problem becomes observation independent.

A factored, n-agent Dec-MDP is said to be *reward independent* if

$$R(s, \mathbf{a}) = f(R_1(s_i, a_i), \ldots, R_n(s_i, a_i)), \tag{7.6}$$

with the constraint that f is some monotonically non-decreasing function. Such a function has the property that $x_i \leq x'_i$ if and only if

$$f(x_1, \ldots, x_i, \ldots, x_n) \leq f(x_1, \ldots, x'_i, \ldots, x_n). \tag{7.7}$$

Table 7.1 Complexity of finite-horizon Dec-MDP subclasses.

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

Under this assumption, the global reward is maximized by maximizing local rewards. Additive local rewards are often used in reward independent models, where

$$R(s,\mathbf{a}) = R_0(s_0) + \sum_i R_i(s_i, a_i). \tag{7.8}$$

Table 7.1 shows the complexity of different subclasses of Dec-MDPs. The simplest case results from having independent transitions, observations, and rewards. It is straightforward to see that, in this case, the problem can be decomposed into n separate MDPs and their solution can then be combined. When only the transitions and observations are independent, the problem becomes NP-complete. Intuitively, the NP-completeness is because the other agents' policies do not affect an agent's state (only the reward attained at the set of local states). Because independent transitions and observations imply local full observability, an agent's observation history does not provide any additional information about its own state—it is already known. Similarly, an agent's observation history does not provide any additional information about the other agents' states because they are independent. As a result, optimal policies become mappings from local states to actions instead of mappings from observation histories (or local state histories as local states are locally fully observable in this case) to actions. All other combinations of independent transitions, observations, and rewards do not reduce the complexity of the problem, leaving it NEXP-complete in the worst case.

7.3.2 ND-POMDPs

A *networked distributed POMDP* (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 or *hypergraph*. A hypergraph is a generalization of a graph in which an edge can 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 upon the state and action components to which the edge connects. The reward function in an ND-POMDP is

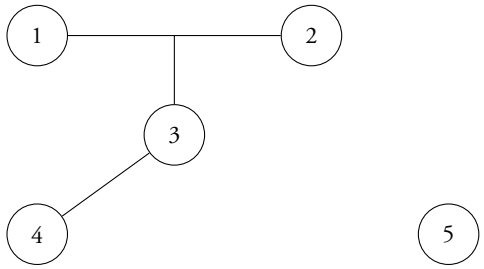


Figure 7.6 Example ND-POMDP structure.

simply the sum of the reward components associated with the edges. This fact allows the value function to be factorable in the same way, and solution methods can take advantage of this additional structure.

Figure 7.6 shows 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. The reward function decomposes as follows:

$$R_{123}(s_1, s_2, s_3, a_1, a_2, a_3) + R_{34}(s_3, s_4, a_3, a_4) + R_5(s_5, a_5). \tag{7.9}$$

Motivating domains for ND-POMDPs are typically sensor network and target tracking problems.

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) and the worst-case complexity remains the same as a full Dec-POMDP (NEXP-complete). While the worst-case complexity remains the same as Dec-POMDPs, algorithms for ND-POMDPs are typically much more scalable in the number of agents. Scalability can increase as the hypergraph becomes less connected.

7.3.3 MMDPs

Another notable subclass is the *multiagent Markov decision process* (MMDP). In an MMDP, each agent is able to observe the true state, making the problem fully observable. Because each agent is able to observe the true state, an MMDP can be solved as an MDP (using some coordination mechanisms to ensure policies are consistent with each other) in polynomial time. An example based on the robot navigation problem

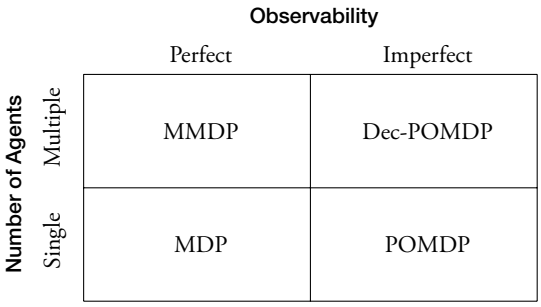


Figure 7.7 The relationship between Dec-POMDPs and some other single and multiagent models.

above for the MMDP case would assume that each robot knows the location of the other robots at each step of the problem. The relationship between MMDPs, Dec-POMDPs, and single-agent models is shown in Figure 7.7.

7.4 Exact Solution Methods

This section discusses optimal algorithms for finite-horizon problems and ϵ-optimal algorithms for infinite-horizon problems. Finite-horizon methods can be used to solve infinite-horizon problems within any ϵ by using a horizon that is large enough to cause further action selection to contribute very little to the overall value. POMDP methods are often scalable enough to produce such solutions, but Dec-POMDP methods are not. As a result, specific infinite-horizon methods for Dec-POMDPs, which use finite-state controllers as solution representations, are discussed. Most solution methods assume the problem is solved centrally offline to produce policies that can be executed online in a decentralized manner. These methods can also produce solutions in a decentralized manner with proper coordination mechanisms for the given algorithm.

7.4.1 Dynamic Programming

Algorithm 7.1 is a dynamic programming algorithm for optimally solving a finite-horizon Dec-POMDP. This approach constructs a set of trees for each agent (represented as π) from the last step until the first. At each step, the algorithm exhaustively generates all next step policies, evaluating them, and then pruning policies that are provably suboptimal for each agent i in turn until no more trees can be removed. This generation, evaluation, and pruning continues until the desired horizon T is reached. Dynamic programming in Dec-POMDPs is similar to the value iteration approach for POMDPs, except generalized belief states are used, resulting in a more complicated pruning step.

Algorithm 7.1 Dynamic programming for Dec-POMDPs

```
1: function DecDynamicProgramming(T)
2:     t ← 0
3:     π_t ← ∅
4:     repeat
5:         π_{t+1} ←ExhaustiveBackup(π_t)
6:         Compute V^{π_{t+1}}
7:         repeat
8:             π̂_{t+1} ← π_{t+1}
9:             for i ∈ I
10:                π̂_{t+1} ←Prune(π̂_{t+1}, i)
11:                Compute V^{π̂_{t+1}}
12:        until |π_t| = |π̂_t|
13:        t ← t + 1
14:    until t = T
15:    return π_t
```

In the dynamic programming algorithm, a set of T-step policy trees, one for each agent, is generated from the bottom up. More precisely, on the last step of the problem, each agent will perform just a single action, which can be represented as a 1-step policy tree. All possible actions for each agent are considered, and each combination of these 1-step trees is evaluated at each state of the problem by using Equation (7.1). Any action that has lower value than some other action for all states and possible actions of the other agents (the generalized belief space) is then pruned. Then, all 2-step policies are generated for each agent by an *exhaustive backup* of the current trees. That is, for each action and each resulting observation, some 1-step tree is chosen. If an agent has $|Q_i|$ 1-step trees, $|A_i|$ actions, and $|\Omega_i|$ observations, there will be $|A_i||Q_i|^{|\Omega_i|}$ 2-step trees. After this exhaustive backup of next step trees is completed for each agent, pruning is again used to reduce their number. This process of backing up trees and pruning continues until horizon T is reached.

The resulting set of trees will contain an optimal solution for horizon T and any initial state of the system. This is because we considered all possible trees at each step and removed only those that were not useful no matter what policies the other agents chose at that step. This conservative pruning of trees ensures that we can safely remove trees that will be suboptimal at a given step.

The linear program used to determine whether a tree can be pruned can be represented as follows. For agent i's given tree q_i and variables ϵ and $x(\mathbf{q}_{-i}, s)$, maximize

ϵ, given

$$\sum_{\mathbf{q}_{-i},s} x(\mathbf{q}_{-i},s)U(\hat{q}_i,\mathbf{q}_{-i},s)+\epsilon \leq \sum_{\mathbf{q}_{-i},s} x(\mathbf{q}_{-i},s)U(q_i,\mathbf{q}_{-i},s) \quad \forall\ \hat{q}_i \tag{7.10}$$

$$\sum_{\mathbf{q}_{-i},s} x(\mathbf{q}_{-i},s) = 1 \text{ and } x(\mathbf{q}_{-i},s) \geq 0 \quad \forall \mathbf{q}_{-i},s. \tag{7.11}$$

This linear program determines if agent i tree, q_i is dominated by comparing its value to other trees for that agent, $\hat{q}_i$. The variable $x(\mathbf{q}_{-i},s)$ is a distribution over trees of the other agents and system states (the generalized belief state). We maximize ϵ while ensuring the variable x that represents the generalized belief state remains a proper probability distribution and test to see if there is some distribution of trees that has higher or equal value for all states and policies of the other agents. Because there is always an alternative with at least equal value, regardless of system state and other agent policies, a tree q_i can be pruned if ϵ is nonpositive.

The test for dominance is used for trees of a given horizon, ensuring that we consider all possible policies for a given horizon and remove those that are not useful no matter what policies are chosen by the other agents. If policies are removed, then the generalized belief space becomes smaller for the other agents (due to the removal of a possible policy to consider), and more policies may be able to be pruned. Thus, we can keep testing for dominated policies for each agent until no agent is able to prune any further policies. Lines 7–11 in Algorithm 7.1 show that pruning continues for all agents while any agent is able to remove any tree.

Unlike value iteration for POMDPs, the policy trees must be retained because it is no longer possible to recover the policy from the value function. Even with one-step lookahead in the POMDP case, we must calculate the belief state after an action is chosen. Because it is not possible to calculate a belief state in the Dec-POMDP case, and because the actions must be chosen based on local information, the optimal value function is not sufficient to generate a Dec-POMDP policy.

7.4.2 Heuristic Search

Instead of computing the policy trees from the bottom up as is done by dynamic programming methods, we can construct the trees from the top down starting from a known initial state. This is the approach of multiagent A* (MAA*), which is an optimal algorithm built on heuristic search techniques. The search is conducted by using an upper bound on the value of partially defined policies (using POMDP or MDP solutions) and then choosing the partial joint policy to expand in a best-first ordering. Actions are then added and an upper bound on this new partial joint policy is found. Again, the highest-valued partial joint policy is chosen and another action choice is fixed. This process continues until a fully defined joint policy is found that

has value that is higher than any of the partial joint policies. Algorithm 7.2 outlines the approach.

Algorithm 7.2 Multiagent A*

```
1:  function MultiagentA*(T, b_0)
2:      V ← −∞
3:      L ← ×_i A_i
4:      repeat
5:          δ ← select(L)
6:          Δ' ← expand(δ)
7:          Δ^T ← fullPols(Δ')
8:          π ← bestFullPol(Δ^T)
9:          v ← valueOf(π)
10:         if v > V
11:             π* ← π
12:             V ← v
13:             prune(L, V)
14:         Δ' ← Δ' \ Δ^T
15:         L ← L \ δ ∪ Δ'
16:     until L is empty
17:     return π*
```

We can grow a joint policy in a top-down fashion by first considering the possible actions each agent can take to begin its policies. MAA* considers all possible combinations of actions that could be taken at that step and constructs a search tree that has these combinations as separate search nodes. In the worst case, a search tree could be constructed that contains all possible policies for each agent by considering all possible combinations of actions at the first step, followed by all possible combinations of actions for each observation at the second step, and so on. As many of these policies may be suboptimal, a more intelligent approach for determining which choices to make is desired.

To assist in choosing better actions, MAA* incorporates a heuristic value for continuing execution for a given number of steps after a partial policy has been executed. That is, using a search based on the A* heuristic search method, we can estimate the value of a set of horizon T policies from a set of horizon t trees using the value of those trees up to horizon t calculated with Equation (7.1) and then some heuristic value for the value of continuing to horizon T.

More formally, we will call some set of horizon $t < T$ policy trees $\mathbf{q}$ a *partial policy* and a set of policies Δ^{T-t} a *completion policy*. A completion policy consists of appending $T-t$ policies to each leaf (last action) in the partial policy. Given a partial

policy and a completion policy, we could evaluate them at a state s as follows:

$$U(\{\mathbf{q}^t, \Delta^{T-t}\}, s) = U(\mathbf{q}^t, s) + U(\Delta^{T-t} \mid \mathbf{q}^t, s). \tag{7.12}$$

Rather than explicitly considering completion policies to append, we can instead estimate the value a completion policy would produce. Therefore, we can produce an estimated value $\hat{U}$ that a partial policy has for the full horizon T as

$$\hat{U}(\mathbf{q}^t, s) = U(\mathbf{q}^t, s) + \hat{U}^{T-t}_{\mathbf{q}^t, s}, \tag{7.13}$$

where $\hat{V}^{T-t}_{\mathbf{q}^t, s}$ is the estimate for the value of continuing until horizon T after executing $\mathbf{q}$ starting at s.

There are many ways we can calculate $V^{T-t}_{\mathbf{q}^t, s}$, but to ensure an optimal policy is produced, we require the estimated value to be at least as high as the optimal value of continuing ($\hat{V} \geq V^*$). MAA* considers relaxing problem assumptions by using MDP or POMDP policies to produce the heuristic values. $V^*_{\text{POMDP}}(b)$ can be defined as the optimal value of a POMDP policy starting at belief b (i.e., a centralized solution in which all observations for all agents are known and actions can be chosen based on this centralized information). Similarly, $V^*_{\text{MDP}}(s)$ can be defined as the optimal value of an MDP policy starting at state s (i.e., a centralized policy assuming the state of the problem can be seen by all agents for the remainder of the problem). It can be shown that $V^*_{\text{Dec-POMDP}} \leq V^*_{\text{POMDP}} \leq V^*_{\text{MDP}}$, which intuitively holds because policies are less constrained as more information is available to the agents. MAA* then evaluates a partial policy by evaluating the policy until its given horizon t and then assumes either the belief state (in the V^*_{POMDP} case) or the state (in the V^*_{MDP} case) is known to the agents thereafter starting from the leaf nodes of $\mathbf{q}^t$.

The search in MAA* then chooses to grow the partial policy with the highest heuristic value. Growing a policy appends actions for each possible observation after the leaves of the current policy. The expansion of this search node adds an exponential number of nodes to the search tree at each step. Each of these new policies is then evaluated to produce an updated heuristic value.

A lower bound on the value of an optimal joint policy is also maintained. If growing a policy results in a horizon T policy, the value of this policy is compared with the lower bound, updating it if the value of this policy is greater. Subsequently, any partial policy with estimated value less than the lower bound can then be pruned. This pruning can occur because the estimated value of a policy is an upper bound on its true value, indicating that it will never have value higher than the policy that produced the lower bound. The search completes when there are no more partial policies to expand. In this case, a set of $\mathbf{q}^T$ policies has been generated with higher value than the heuristic values of any partial policy.

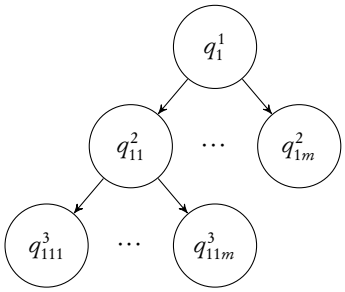


Figure 7.8 An example search tree in MAA*.

We now describe Algorithm 7.2 in more detail. A lower bound value, $\underline{V}$, which represents the value of the best known joint policy is initialized to negative infinity. An open list L, which represents the partial policies that are available to be expanded, is initialized to all joint actions for the agents. At each step, the partial joint policy (node in the search tree) with the highest estimated value $\hat{V}$ is selected. Then, this partial policy is expanded, generating all next step policies for that joint policy (all children in the search tree). This set is called Δ'. The set of full horizon T joint policies is now gathered into Δ^T, each of which is evaluated and the one with the highest value v is chosen. If this value is higher than the value of the best known full policy, the lower bound value and pointer to best policy are updated. Also, any partial policy in the open list that has estimated upper bound value (using the Q_{MDP} or Q_{POMDP} heuristics) lower than $\underline{V}$ is pruned. The full trees are removed from the set of expanded nodes and the selected node is removed from the open list. The remaining expanded nodes are then added to the open list. This algorithm continues until the open list is empty and returns an optimal joint policy π^*.

An example of this search is shown in Figure 7.8. Each search node represents a joint policy for a given horizon. Subscripts represent the indices of the search node and each of the ancestor nodes in the search tree, while superscripts represent the horizon of the joint policy. Ancestors are indexed to signify that partial policies are shared for the appropriate horizon. Note that m represents the number of possible next-step joint policies, which is $|A_{max}|^{n|\Omega_{max}|}$, given the largest action and observation sets amoung the n agents A_{max}, Ω_{max}. The available partial policies (the open list) are represented by the leaves of the tree while the nodes that have already been expanded are shown as internal nodes.

Algorithm 7.3 Policy iteration for Dec-POMDPs

```
1:  function DecPolicyIteration(π_0, ε)
2:      t ← 0
3:      repeat
4:          π_{t+1} ← ExhaustiveBackup(π_t)
5:          Compute V^{π_{t+1}}
6:          repeat
7:              π̂_{t+1} ← π_{t+1}
8:              for i ∈ I
9:                  π̂_{t+1} ← Prune(π̂_{t+1}, i)
10:                 UpdateController(π̂_{t+1}, i)
11:                 Compute V^{π̂_{t+1}}
12:         until |π_t| = |π̂_t|
13:         t ← t + 1
14:     until γ^{t+1}|R_max| / (1−γ) ≤ ε
15:     return π_t
```

7.4.3 Policy Iteration

Because the infinite-horizon problem is undecidable, it may not be possible to generate a solution with the exact optimal value. As a consequence, methods focus on producing solutions within ϵ of the optimal value.

The policy iteration approach for Dec-POMDPs is similar to the finite-horizon dynamic programming algorithm, except finite-state controllers are used as policy representations (like the policy iteration approach for POMDPs). Starting from an initial controller for each agent, nodes are added at each step by using an exhaustive backup to produce any possible next-step policy for each agent. Pruning can then be completed to remove a node in an agent's controller if it has lower value than beginning in another node for all states of the system and all possible controllers of the other agents. These exhaustive backups and pruning steps continue until the solution is provably within ϵ of an optimal solution. This algorithm can produce an ϵ-optimal policy in a finite number of steps. The details of policy iteration follow.

The policy iteration algorithm is shown in Algorithm 7.3. The input is an initial joint controller, π_0 and a parameter ϵ. At each step, evaluation, backup and pruning occurs. The controller is evaluated using Equation (7.2). Next, an exhaustive backup is performed to add nodes to the local controllers. An exhaustive backup adds nodes to the local controllers for all agents at once. Similar to the finite-horizon case, for each agent i, $|A_i||Q_i|^{|\Omega_i|}$ nodes are added to the local controller, one for each one-step policy. Note that repeated application of exhaustive backups amounts to a brute force

search in the space of deterministic policies. Continuing these exhaustive backups converges to optimality, but is obviously quite inefficient.

To increase the efficiency of the algorithm, pruning takes place. Recall that planning takes place offline, so the controllers for each agent are known at each step, but agents will not know which node of their controller any of the other agents will be in during execution. As a result, pruning must be completed over the generalized belief space (using a linear program that is very similar to that described for finite-horizon dynamic programming). That is, a node for an agent's controller can only be pruned if there is some combination of nodes that has higher value for all states of the system and at all nodes of the other agents' controllers. If this condition holds, edges to the removed node are then redirected to the dominating nodes. Because a node may be dominated by a distribution of other nodes, the resulting transitions may be stochastic rather than deterministic. The updated controller is evaluated, and pruning continues until no agent can remove any further nodes.

In contrast to the single-agent case, there is no Bellman residual for testing convergence to ϵ-optimality. We resort to a simpler test based on the discount rate and the number of iterations so far. Let $|R_{\max}|$ be the largest absolute value of an immediate reward possible in the Dec-POMDP. The algorithm terminates after iteration t if $\frac{\gamma^{t+1}|R_{\max}|}{1-\gamma} \leq \epsilon$. At this point, due to discounting, the value of any policy after step t is less than ϵ.

7.5 Approximate Solution Methods

In this section, we discuss approximate dynamic programming methods for finite-horizon problems and fixed-size controller-based methods for infinite-horizon problems. The algorithms in this section do not possess error bounds.

7.5.1 Memory Bounded Dynamic Programming

The major limitation of dynamic programming approaches is the explosion of memory and time requirements as the horizon grows. This explosion occurs because each step requires generating and evaluating all joint policy trees (sets of policy trees for each agent) before performing the pruning step. Approximate dynamic programming techniques can mitigate this problem by keeping a fixed number of policy trees for each agent at each step, using a parameter called *MaxTrees*. This approach, called *memory-bounded dynamic programming* (MBDP), is outlined in Algorithm 7.4.

MBDP merges top-down (heuristic search) and bottom-up (dynamic programming) approaches by using heuristics (referred to as H in the algorithm) to choose top-down policies for each agent up to a given horizon. That is, dynamic programming proceeds as before, but after each backup, *MaxTrees* belief states are generated

Algorithm 7.4 Memory-bounded dynamic programming (MBDP)

function MBDP(*MaxTrees*,T,H)
  $t \leftarrow 0$
  $\pi_t \leftarrow \emptyset$
  repeat
    $\pi_{t+1} \leftarrow$ExhaustiveBackup(π_t)
    Compute $V^{\pi_{t+1}}$
    $\hat{\pi}_{t+1} \leftarrow \emptyset$
    for $k \in$ *MaxTrees*
      $b_k \leftarrow$ GenerateBelief(H,$T-t-1$)
      $\hat{\pi}_{t+1} \leftarrow \hat{\pi}_{t+1} \cup \arg\max_{\pi_{t+1}} V^{\pi_{t+1}}(b_k)$
    $t \leftarrow t+1$
    $\pi_{t+1} \leftarrow \hat{\pi}_{t+1}$
  until $t = T$
  return π_t

using heuristics to perform top-down sampling until the current step of dynamic programming is reached, ($T-t$ if dynamic programming is at step t). It is then assumed that the resulting belief states are revealed to the agents, and only the trees that have the highest value at these belief states are retained. While during execution, the belief state will not truly be revealed to the agents, the hope is that high-valued decentralized policies can still be produced using this strategy. MBDP is conducted in an iterative fashion similar to traditional dynamic programming. For example, in a problem of horizon T, heuristic policies can be used for the first $T-1$ steps and dynamic programming can find the best 1-step trees (actions) for the resulting beliefs. Then, heuristic policies can be used for the first $T-2$ steps and the *MaxTrees* 1-step trees can be built up to 2 steps by dynamic programming. This process continues until *MaxTrees* horizon T trees have been constructed. Heuristics that have been used include MDP and random policies. Because only a fixed number of trees are retained at each step, the result is a suboptimal, but much more scalable algorithm. In fact, since the number of policies retained at each step is bounded by *MaxTrees*, MBDP has time and space complexity linear in the horizon.

7.5.2 Joint Equilibrium Search

As an alternative to MBDP-based approaches, a method called joint equilibrium search for policies (JESP) utilizes alternating best response. The JESP algorithm is shown in Algorithm 7.5. Initial policies are generated for all agents and then all but one are held fixed. The remaining agent can then calculate a best response (local optimum) to the

Algorithm 7.5 Joint equilibrium search for strategies (JESP) without DP

```
1:  function JESP(π)
2:      k = 0
3:      π^k ← π
4:      repeat
5:          π^{k+1} ← π^k
6:          k ← k + 1
7:          for i ∈ I
8:              Compute V^{π^k}
9:              π^k(i) ← BestResponse(π^k, −i)
10:     until π^k = π^{k−1}
11:     return π^k
```

fixed policies. This agent's policy then becomes fixed, and the next agent calculates a best response. This process continues until no agents change their policies. The result is a policy that is only locally optimal, but it may be high-valued. JESP can be made more efficient by incorporating dynamic programming in the policy generation. Note that JESP can be thought of as finding a Nash equilibria (as discussed in Section 3.3) in the cooperative game represented as the Dec-POMDP.

7.6 Communication

Communication can be implicitly represented using observations, but more explicit representations of communication have also been developed. Free and instantaneous communication is equivalent to centralization because all agents can have access to all observations at each step. When communication is delayed or has a cost, agents must reason about what and when to communicate. The complexity classes of the Dec-POMDP model with different types of observability and communication are shown in Table 7.2.

One extension of the general Dec-POMDP model to explicitly include communication is the decentralized partially observable Markov decision process with communication (Dec-POMDP-Com). A Dec-POMDP-Com is a Dec-POMDP where each agent can send a message at each step. The reward at each step is a function of the joint state, joint action, and the set of messages sent by the agents.

Producing an optimal solution of a Dec-POMDP-Com has the same complexity as solving the general Dec-POMDP model (NEXP-complete), and similar algorithms can be used to solve it as well. It can be shown that the optimal value of communicating an agent's history since the last communication will have value at least as high as any other set of possible messages, assuming fixed communication costs. As a result,

Table 7.2 The effect of communication on complexity of models with different observability.

Observability	General Communication	Free Communication
Full	MMDP (P)	MMDP (P)
Joint Full	Dec-MDP (NEXP)	MMDP (P)
Partial	Dec-POMDP (NEXP)	MPOMDP (PSPACE)

many communication approaches assume that observation histories are used during communication. Many methods assume communication decisions are made by each agent separately, but some models assume agents can force all others to send their observation histories (allowing a POMDP belief state to be calculated).

A natural approach to solving a Dec-POMDP with communication is to produce a centralized (POMDP) plan and communicate when the observations received by an agent would cause it to choose a different action from the one prescribed by the centralized plan. This approach can be thought of as the agents agreeing on a policy before execution, and when it is noticed (by an agent's local observations) that this policy could be improved, communication takes place. This approach does not explicitly consider communication cost or delay, but can limit the number of times communication takes place.

7.7 Summary

- Dec-POMDPs can represent cooperative mutltiagent problems with action outcome uncertainty, observation uncertainty, and costly, lossy, or no communication.
- Like MDPs and POMDPs, Dec-POMDPs model sequential problems using a decision-theoretic approach that utilizes probabilities for uncertainties and values for outcomes.
- Unlike the single-agent models, each agent must make choices based solely on its own observation history.
- Solving finite-horizon Dec-POMDPs is NEXP-complete.
- Many subclasses of Dec-POMDPs exist that are more efficient in theory or practice.
- Algorithms have been developed that can produce optimal or ϵ-optimal solutions for finite and infinite-horizon Dec-POMDPs.
- More scalable approximate algorithms have also been developed.
- Communication can also be explicitly modeled to assist in determining when and what to communicate to improve performance.

7.8 Further Reading

General overviews of Dec-POMDPs with additional algorithms and models are provided by Seuken and Zilberstein [1], Oliehoek [2], and Goldman and Zilberstein [3]. The complexity of the general Dec-POMDP was proven by Bernstein et al. [4]. A model very similar to a Dec-POMDP is the multiagent team decision problem (MTDP) [5]. Finite-horizon dynamic programming is presented by Hansen, Bernstein, and Zilberstein [6] and infinite-horizon policy iteration is described by Bernstein et al. [7]. Multiagent A* is presented by Szer, Charpillet, and Zilberstein [8].

Dec-MDPs with independent transitions and observations were first discussed and solved by Becker et al. [9]. Dec-MDPs with event-driven interactions, considering limited transition dependence, were also discussed [10]. Allen and Zilberstein discussed a number of different modeling assumptions and resulting complexity [11]. ND-POMDPs were first proposed by Nair et al. [12], and MMDPs are presented by Boutilier [13]. Approximate finite-horizon algorithms MBDP and JESP are described by Seuken and Zilberstein [14] and Nair et al. [15], respectively. Approximate infinite-horizon algorithms can be found in the literature [16]–[18].

The Dec-POMDP-Com model was presented by Goldman and Zilberstein [3]. The idea of using a centralized policy as a basis for communication was described by Roth, Simmons, and Veloso [19]. Forced synchronizing communication was discussed by Nair and Tambe [20].

Optimal finite-horizon algorithms have been improved in a few directions. Pruning can be used while conducting the backup to limit the subtrees that need to be considered [21]. Other work has also compressed policies rather than agent histories, improving the efficiency of the linear program used for pruning [22]. MAA* has been improved in several ways, including the incorporation of new heuristics and improved search [23]–[25]. Also, recently, two approaches have been developed for generating more concise sufficient statistics for offline planning, proving the sufficiency of distributions of states and joint histories [26] and converting Dec-POMDPs into POMDPs using decentralizable policies [27].

Approximate algorithms have also seen additional improvements. A number of approaches have improved upon MBDP by compressing observations [28], replacing the exhaustive backup with a branch-and-bound search in the space of joint policy trees [29], as well as constraint optimization [30] and linear programming [31] to scale up the selection of the trees at each step. Additional fixed-size controller approaches have been developed that utilize an alternative representation of the controller as a Mealy machine to increase performance [32], employ expectation maximization for parameter optimization [33], or make use of more structured periodic controllers and an improved search technique [34].

Algorithms for subclasses have also been developed. The algorithm for solving transition and observation independent Dec-MDPs was the coverage set method [9]. Additional methods have also been developed to solve independent transition and observation Dec-MDPs more efficiently. These methods include a bilinear programming algorithm [35], a mixture of heuristic search and constraint optimization [36], and recasting the problem as a continuous MDP [37]. Optimal and approximate methods for solving ND-POMDPs have been proposed [12]. Other ND-POMDP methods have also been developed, such as those producing quality bounded solutions [38] and using finite-state controllers for agent policies [39].

There are other models of communication discussed in the literature [3], [5]. Communication has been studied in the context of locally fully observable Dec-MDPs with independent transitions and observations. This problem can be modeled with a cost of communication to receive the local states of the other agents and solved with a set of heuristic approaches [40]. Myopic communication, where an agent decides whether or not to communicate based on the assumption that communication can take place on this step or never again, has also been shown to work well in many cases [41]. Other types of communication explored include stochastically delayed communication [42] and communication for online planning in Dec-POMDPs [43].

Factored models have been studied in the context of Dec-POMDPs. General factored models have been described and solved [44]. Witwicki and Durfee summarized different types of factored models and their complexity [45], and improved algorithms have been developed [46].

In general, the research community has focused on planning methods for models that are similar to Dec-POMDPs, but a few learning methods have been explored. These include model-free reinforcement learning methods using gradient-based methods to improve the policies [47], [48] and using communication to learn solutions in ND-POMDPs [49].

Additional solution methods for general Dec-POMDPs have also been proposed. These include a mixed integer linear programming approach for Dec-POMDPs [50] and sampling methods [51], [52].

A number of applications have been studied, including multi-robot coordination in the form of space exploration rovers [53], helicopter flights [5] and navigation [54]–[56], load balancing for decentralized queues [57], network congestion control [58], multiaccess broadcast channels [59], network routing [60], sensor network management [12], target tracking [12], [61], and weather phenomena [62].

References

1. S. Seuken and S. Zilberstein, "Formal Models and Algorithms for Decentralized Control of Multiple Agents," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 2, pp. 190–250, 2008.
2. F.A. Oliehoek, "Decentralized POMDPs," in *Reinforcement Learning: State of the Art*, M. Wiering and M. van Otterlo, eds., vol. 12, Springer, 2012.
3. C.V. Goldman and S. Zilberstein, "Decentralized Control of Cooperative Systems: Categorization and Complexity Analysis," *Journal of Artificial Intelligence Research*, vol. 22, pp. 143–174, 2004.
4. D.S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The Complexity of Decentralized Control of Markov Decision Processes," *Mathematics of Operations Research*, vol. 27, no. 4, pp. 819–840, 2002.
5. D.V. Pynadath and M. Tambe, "The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models," *Journal of Artificial Intelligence Research*, vol. 16, pp. 389–423, 2002.
6. E.A. Hansen, D.S. Bernstein, and S. Zilberstein, "Dynamic Programming for Partially Observable Stochastic Games," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2004.
7. D.S. Bernstein, C. Amato, E.A. Hansen, and S. Zilberstein, "Policy Iteration for Decentralized Control of Markov Decision Processes," *Journal of Artificial Intelligence Research*, vol. 34, pp. 89–132, 2009.
8. 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.
9. R. Becker, S. Zilberstein, V. Lesser, and C.V. Goldman, "Solving Transition-Independent Decentralized Markov Decision Processes," *Journal of Artificial Intelligence Research*, vol. 22, pp. 423–455, 2004.
10. R. Becker, V. Lesser, and S. Zilberstein, "Decentralized Markov Decision Processes with Event-Driven Interactions," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2004.
11. M. Allen and S. Zilberstein, "Complexity of Decentralized Control: Special Cases," in *Advances in Neural Information Processing Systems (NIPS)*, 2009.
12. R. Nair, P. Varakantham, M. Tambe, and M. Yokoo, "Networked Distributed POMDPS: a Synthesis of Distributed Constraint Optimization and POMDPs," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2005.
13. C. Boutilier, "Sequential Optimality and Coordination in Multiagent Systems," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

14. S. Seuken and S. Zilberstein, "Memory-Bounded Dynamic Programming for DEC-POMDPs," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.
15. R. Nair, D. Pynadath, M. Yokoo, M. Tambe, and S. Marsella, "Taming Decentralized POMDPS: Towards Efficient Policy Computation for Multiagent Settings," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.
16. D. Szer and F. Charpillet, "An Optimal Best-First Search Algorithm for Solving Infinite Horizon DEC-POMDPs," in *European Conference on Machine Learning (ECML)*, 2005.
17. D.S. Bernstein, E.A. Hansen, and S. Zilberstein, "Bounded Policy Iteration for Decentralized POMDPs," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.
18. C. Amato, D.S. Bernstein, and S. Zilberstein, "Optimizing Fixed-Size Stochastic Controllers for POMDPs and Decentralized POMDPs," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 21, no. 3, pp. 293–320, 2010.
19. M. Roth, R. Simmons, and M.M. Veloso, "Reasoning About Joint Beliefs for Execution-Time Communication Decisions," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2005.
20. R. Nair and M. Tambe, "Communication for Improving Policy Computation in Distributed POMDPs," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2004.
21. C. Amato, J.S. Dibangoye, and S. Zilberstein, "Incremental Policy Generation for Finite-Horizon DEC-POMDPs," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.
22. A. Boularias and B. Chaib-draa, "Exact Dynamic Programming for Decentralized POMDPs with Lossless Policy Compression," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2008.
23. F.A. Oliehoek, M.T.J. Spaan, and N. Vlassis, "Optimal and Approximate Q-Value Functions for Decentralized POMDPs," *Journal of Artificial Intelligence Research*, vol. 32, pp. 289–353, 2008.
24. F.A. Oliehoek, M.T.J. Spaan, and C. Amato, "Scaling Up Optimal Heuristic Search in Dec-POMDPs via Incremental Expansion," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.
25. F.A. Oliehoek, M.T.J. Spaan, C. Amato, and S. Whiteson, "Incremental Clustering and Expansion for Faster Optimal Planning in Dec-POMDPs," *Journal of Artificial Intelligence Research*, vol. 46, pp. 449–509, 2013.
26. F.A. Oliehoek, "Sufficient Plan-Time Statistics for Decentralized POMDPs," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

27. J.S. Dibangoye, C. Amato, O. Buffet, and F. Charpillet, "Optimally Solving Dec-POMDPs as Continuous-State MDPs," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

28. A. Carlin and S. Zilberstein, "Value-Based Observation Compression for DEC-POMDPs," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.

29. J.S. Dibangoye, A.-I. Mouaddib, and B. Chaib-draa, "Point-Based Incremental Pruning Heuristic for Solving Finite-Horizon DEC-POMDPs," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.

30. A. Kumar and S. Zilberstein, "Point-Based Backup for Decentralized POMDPs: Complexity and New Algorithms," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.

31. F. Wu, S. Zilberstein, and X. Chen, "Point-Based Policy Generation for Decentralized POMDPs," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.

32. C. Amato, B. Bonet, and S. Zilberstein, "Finite-State Controllers Based on Mealy Machines for Centralized and Decentralized POMDPs," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

33. A. Kumar and S. Zilberstein, "Anytime Planning for Decentralized POMDPs using Expectation Maximization," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2010.

34. J.K. Pajarinen and J. Peltonen, "Periodic Finite State Controllers for Efficient POMDP and DEC-POMDP Planning," in *Advances in Neural Information Processing Systems (NIPS)*, 2011.

35. M. Petrik and S. Zilberstein, "A Bilinear Programming Approach for Multiagent Planning," *Journal of Artificial Intelligence Research*, vol. 35, pp. 235–274, 2009.

36. J.S. Dibangoye, C. Amato, and A. Doniec, "Scaling Up Decentralized MDPs Through Heuristic Search," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2012.

37. J.S. Dibangoye, C. Amato, A. Doniec, and F. Charpillet, "Producing Efficient Error-Bounded Solutions for Transition Independent Decentralized MDPs," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2013.

38. P. Varakantham, J. Marecki, Y. Yabu, M. Tambe, and M. Yokoo, "Letting Loose a SPIDER on a Network of POMDPS: Generating Quality Guaranteed Policies," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2007.

39. J. Marecki, T. Gupta, P. Varakantham, M. Tambe, and M. Yokoo, "Not All Agents Are Equal: Scaling up Distributed POMDPs for Agent Networks," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.

40. P. Xuan and V. Lesser, "Multi-Agent Policies: From Centralized Ones to Decentralized Ones," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2002.

41. R. Becker, A. Carlin, V. Lesser, and S. Zilberstein, "Analyzing Myopic Approaches for Multi-Agent Communication," *Computational Intelligence*, vol. 25, no. 1, pp. 31–50, 2009.

42. M.T.J. Spaan, F.A. Oliehoek, and N. Vlassis, "Multiagent Planning Under Uncertainty with Stochastic Communication Delays," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2008.

43. F. Wu, S. Zilberstein, and X. Chen, "Multi-Agent Online Planning with Communication," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.

44. F.A. Oliehoek, M.T.J. Spaan, S. Whiteson, and N. Vlassis, "Exploiting Locality of Interaction in Factored Dec-POMDPs," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.

45. S.J. Witwicki and E.H. Durfee, "Towards a Unifying Characterization for Quantifying Weak Coupling in Dec-POMDPs," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

46. S.J. Witwicki, F.A. Oliehoek, and L.P. Kaelbling, "Heuristic Search of Multiagent Influence Space," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2012.

47. A. Dutech, O. Buffet, and F. Charpillet, "Multi-Agent Systems by Incremental Gradient Reinforcement Learning," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.

48. L. Peshkin, K.-E. Kim, N. Meuleau, and L.P. Kaelbling, "Learning to Cooperate via Policy Search," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000.

49. C. Zhang and V.R. Lesser, "Coordinated Multi-Agent Reinforcement Learning in Networked Distributed POMDPs," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2011.

50. R. Aras, A. Dutech, and F. Charpillet, "Mixed Integer Linear Programming For Exact Finite-Horizon Planning In Decentralized POMDPs," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2007.

51. C. Amato and S. Zilberstein, "Achieving Goals in Decentralized POMDPs," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.

52. F.A. Oliehoek, J.F. Kooi, and N. Vlassis, "The Cross-Entropy Method for Policy Search in Decentralized POMDPs," *Informatica*, vol. 32, pp. 341–357, 2008.

53. D.S. Bernstein, S. Zilberstein, R. Washington, and J.L. Bresina, "Planetary Rover Control as a Markov Decision Process," in *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2001.

54. R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun, "Game Theoretic Control for Robot Teams," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

55. M.T.J. Spaan and F.S. Melo, "Interaction-Driven Markov Games for Decentralized Multiagent Planning Under Uncertainty," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.

56. L. Matignon, L. Jeanpierre, and A.-I. Mouaddib, "Coordinated Multi-Robot Exploration Under Communication Constraints Using Decentralized Markov Decision Processes," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

57. R. Cogill, M. Rotkowitz, B. Van Roy, and S. Lall, "An Approximate Dynamic Programming Approach to Decentralized Control of Stochastic Systems," in *Allerton Conference on Communication, Control, and Computing*, 2004.

58. K. Winstein and H. Balakrishnan, "TCP Ex Machina: Computer-Generated Congestion Control," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2013.

59. J.M. Ooi and G.W. Wornell, "Decentralized Control of a Multiple Access Broadcast Channel: Performance Bounds," in *35th Conference on Decision and Control*, 1996.

60. L. Peshkin and V. Savova, "Reinforcement Learning for Adaptive Routing," in *International Joint Conference on Neural Networks*, 2002.

61. A. Kumar and S. Zilberstein, "Constraint-Based Dynamic Programming for Decentralized POMDPs with Structured Interactions," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.

62. ——, "Event-Detecting Multi-Agent MDPS: Complexity and Constant-Factor Approximation," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2009.