# Chapter 11

# Multiagent Planning, Control, and Execution

*Ed Durfee and Shlomo Zilberstein*

## 1 Introduction

Planning is important to an agent because its current and upcoming choices of actions can intentionally establish, or accidentally undo, the conditions that later actions depend upon to reach desirable states of the world. Hence, planning in single-agent systems is concerned with how an agent can efficiently model and select from alternative sequences of actions, preferably without considering every possible sequence. In a multiagent setting, the added complication is that decisions an agent makes about near-term actions can impact the future actions that *other agents* can (or cannot) take. Similarly, knowing what actions other agents plan to take in the future could impact an agent's current action choices. And, unlike single agents, multiple agents can act concurrently. Therefore an agent's choice of action at any given time can impact and be impacted by the action choices of other agents at the same time. Because the space of possible joint courses of action the agents could take grows exponentially with the number of agents (as we will detail later), planning in a multiagent world is inherently intractable, a problem that is compounded in dynamic, partially-observable, and/or non-deterministic environments. Yet, when agents are cooperative, as we will assume in this chapter, then they should strive to make decisions that collectively over time achieve their joint objectives as effectively as possible.

The above paragraph captures in a simplistic way the themes of this chapter. By *multiagent control*, we refer to how agents in a multiagent system can be provided with and utilize information to make better decisions about what to do *now* so that their joint actions can further the achievement of joint objectives. *Multiagent planning* focuses not just on current decisions, but also on sequences of decisions, allowing an agent to "look ahead" so as to establish conditions for another agent that allow it to achieve desired shared goals. From the cooperative perspective, an agent should be willing to incur local cost if by doing so it enables other agents to achieve benefits that more than offset that cost. *Multiagent execution* extends and in some senses combines multiagent planning and control, where agents both proactively plan their (inter)actions to guide the evolution of their shared environment, but also reactively control their behaviors in response to emergent or unlikely events.

In single-agent planning, a key to getting traction on solving hard problems is to exploit structure, typically involving notions of locality and composition. An agent's *state* is typically composed of *features* (for example, propositions about what facts are true in the agent's environment), and an action the agent can take typically involves only a small subset (*locality*) of features. By focusing only on features of interest and the actions that involve them, an agent can focus its search only on a much smaller space of relevant plans. Similarly, as we will see in this chapter, solving multiagent planning and control problems can depend critically on exploiting problem structure, where that structure extends to locality of interacting agents (e.g., an agent's action choices only directly affect a few other agents), and locality of involved features (e.g., few agents can directly affect any particular feature, and/or only a few of the features an agent cares about can be affected by others). Agents can exploit such structure to formulate their joint plans through *composition*: the solutions for the different localities can be combined (relatively) straightforwardly into a comprehensive multiagent solution.

This chapter builds on the topics of the previous chapters to describe the concepts and algorithms that comprise the foundations of multiagent control, planning, and execution. We assume that the reader is already familiar with protocols of interaction; here those protocols are used in the context of coordinating cooperative multiagent action. We also assume the reader is familiar with traditional AI search techniques, planning algorithms and representations, and models for reasoning under uncertainty. We make liberal use of the relevant concepts as we delve into their multiagent analogues.

## 2 Characterizing Multiagent Planning and Control

As with many topics in multiagent systems (and artificial intelligence, and computer science ...), a phrase like "multiagent planning" or "multiagent control" can mean different things to different people. We do not claim that our characterization here is necessarily the consensus opinion of the community, but it should give the reader of this chapter a sense of the problems that are (and are not) within the space of problems considered here.

Multiagent planning is something of an ambiguous term, because it is unclear exactly what is "multiagent." It could be that the operative issue is that, as a consequence of possibly centralized planning, a plan is formulated that can be distributed across and acted upon by a set of agent systems. Alternatively, the operative issue could be that the planning process should itself be multiagent, whether or not the resulting plan(s) are. Or perhaps both issues are of interest.

In this chapter, we consider both multiagent plans and multiagent plan formation as requirements. The case where neither holds is simply traditional single-agent planning. The case where multiple agents cooperatively generate a single-agent plan (such as where a set of planning specialists can contribute to the formation of a plan for the manufacture of an artifact [34], is effectively an instance of cooperative problem solving, where the problem being solved is the construction of a plan. Depending on the nature of the plan being devised, techniques such as distributed constraint satisfaction [67], distributed constraint optimization (Chapter 12), or distributed search [62] can be employed to jointly construct such a plan. Finally, consider the case where a centralized planner builds a detailed collection of plans to distribute among the agents, such that the behavior of each agent is precisely dictated. While beautifully coordinated behavior can ensue, the agents are stripped of any autonomy, and are thus arguably not agents in any interesting sense any more. In essence, this is multi-effector planning, rather than multiagent planning.

If the centralized planner provides less detailed guidance, however, then agents can exercise their "agent" attributes to utilize local awareness of the world, along with local preferences, knowledge, and capabilities, to more autonomously and individually decide on current actions, and even to plan future actions. When this occurs, then the multiagent plan and the plan formation process are both inherently distributed among agents: no single agent forms or even might be aware of the entire joint plan, since different agents may have made their own local plan elaborations and refinements.

What happens if there is no centralized planner, and hence no centralized guidance, at all? This is an interesting question. If we say that the agents could cooperatively converge on an effective (distributed) joint plan without a centralized planner, where did the guidance come from to do so? Typically, some central-

izing entity (an agent, a human system designer, a group of people comprising a standards body) will have devised and disseminated some guidelines, such as interaction plans (aka protocols) and the rules for using them, which the agents count upon to communicate and cooperate with each other. How and whether the environment can itself provide the structure to allow dissimilar agents to converge on cooperative plans for non-trivial problems, or can engender the unguided emergence of languages and protocols that enable cooperation, is beyond the scope of this chapter.

The preceding thus sets the table for our exploration of multiagent planning and control techniques in this chapter. We begin in Section 3 with looking at the process of creating centralized guidelines that push agents to make good control decisions (about current actions) and/or planning decisions (about sequences of actions). As we shall see, in some cases these guidelines can guarantee that the control decisions or plans that each agent makes in adherence to the guidelines *must* be jointly coordinated. In such cases, coordination precedes planning.

We then turn to the opposite (Section 4), when planning precedes coordination, where the guidelines are weaker (typically, more general-purpose) and hence do not constrain the space of joint plans much. Instead, each agent can elaborate its own plan to achieve its assigned goals, and then these plans are coordinated by, for example, adjusting the timing of agents' activities to preclude interference.

Unfortunately, for a variety of interesting problems, including problems where unexpected events can occur at runtime, (local) planning of individual actions and (multiagent) coordination of interactions need to be done together, in an interleaved manner. In Section 5, we look at how such multiagent sequential decision-making problems can be formulated as decentralized (partially-observable) Markov decision processes, and describe techniques for finding optimal and approximately-optimal solutions (joint plans) in such problems.

Finally, finding good joint plans is only useful if agents can successfully execute them. Since planning is done using a model of the environment, and that model might not correctly represent the actual environment at the time the plan is executed, agents should monitor the progress of their plans against expectations, and repair their joint plans in response to deviations. These ideas are familiar (though still challenging) in the single-agent planning world; we conclude this chapter (Section 6) describing strategies to handle similar problems in the multiagent setting.

# 3 Coordination Prior to Local Planning

Developers of distributed systems typically anticipate how entities within a system might need to interact, and predefine interaction plan templates for the en-

tities to fill in and follow. Examples of such interaction plan templates abound in this book. These templates can take the form of interagent protocols, defining the possible sequences of communicative acts between agents, where the content of these acts can be domain dependent. For example, agents solving a distributed constraint satisfaction problem follow protocols for exchanging information about tentative assignments of values to variables, or of no-good assignments that collectively violate constraints [67]. As another example, agents solving a resource allocation problem can work within auction mechanisms that have been designed to cause information exchanges to converge on efficient allocations (Chapter 7).

## 3.1 Social Laws and Conventions

We begin with a simple strategy to ensure sufficient coordination of agents' actions, a strategy that has been characterized as imposing *social laws* on agents [55, 56]. The idea is to identify joint states that should not be allowed to arise, and to impose restrictions on agents' action choices to prevent them.

A canonical application domain for social laws is in coordinating mobile robots. Collisions between robots leads to system degradation (robots become disabled) and cost (robots need repairs), and thus should be avoided. If space is discretized, such as modeling it as a grid, then states where two or more robots are in the same grid coordinate should be prevented. Thus, one law to impose on the robots is that a robot should never move into a neighboring location that is occupied.

A moment's reflection reveals that such a law is insufficient, because it fails to prevent two or more robots from simultaneously entering the same empty location from different directions. One way to strengthen the social law is to prohibit agents from entering a location from more than one direction. If each location in the grid is to be reachable from every other location, this stronger law effectively defines a directed cycle through the grid locations such that each location is visited exactly once.

The stronger social law leads to agents moving through the locations in a sort of "conga line," where each can move to its next location when that location is empty. Agents do not need to coordinate their action choices as they decide where they want to go, because so long as agents obey the law, collisions cannot arise. However, such prebuilt coordination generally comes at a price. An agent might take a very circuitous route to get to a desired destination because of the social law, when it could have potentially gotten where it wanted much more directly and safely because other agents were far away. In human terms, going the wrong way on a one-way street might be more efficient late at night when the odds of encountering oncoming cars is negligible. But deciding when it is safe to break such laws requires agents to reason about (and often communicate with) each other. A

purpose of social laws is to relieve agents of the burden of explicitly coordinating, potentially at the price of some degree of inefficiency in joint behaviors.

The flip side of social laws that tell agents what they are prohibited from doing in certain circumstances is the notion of *conventions*, which tell agents what they should (or must) do. The conceptual framework for conventions is the same as for social laws, which is to identify undesirable joint situations and to constrain agents to actions that avoid them. A canonical application domain for conventions is when agents share joint intentions [33], such that they have committed to work together on achieving some mutually-desired goal. If, in the midst of pursuing this joint goal, an agent comes to believe that the goal is unachievable, then it would be irrational for the agent to continue pursuing it. However, a state in which some agents are continuing to pursue a joint goal while others have dropped it as unachievable is arguably an undesirable state, since the former agents are taking futile actions. Hence, agents in the joint intentions framework follow a convention that they must notify each other if they come to believe the joint goal cannot be achieved.

Other flavors of these concepts have been introduced, such as that agents should return shared resources to their default state after usage (e.g., putting a tool back where it belongs when finished using it) or even go slightly out of their way to make a shared environment more conducive to goal achievement for other agents (e.g., widening a path while following it to make its traversal easier for later agents) [29]. The algorithmic model shared by them all is:

1. Identify joint states that should be avoided (or sought).
2. Work backward through agents' joint actions to identify possible precursor states to these states.
3. Impose constraints on agents' action choices in the precursor states to prevent (or require) joint actions accordingly.

Note that this process can recurse. If a precursor of a state to avoid leads inexorably to the undesirable state, then the precursor state should be added to the states to be avoided, and the algorithm should work backwards from it too. Similarly, if there is a way to go assuredly to a sought state from its precursor, the precursor can be added to the set of sought states.

## 3.2 Organizational Structuring

While social laws and conventions apply equally to all agents, cooperation in some types of problems can be better achieved if agents are differentially biased in the actions they choose to, or choose not to, take. Organizational structures

are a familiar example of this in human institutions. An organizational structure defines, for example, a set of different organizational roles with identified responsibilities, and connections between roles to direct exchanges of information and to dictate authority relationships. A good human organizational structure is one that provides the people occupying each of the roles with guidance about how to prioritize their tasks and direct their communications such that their complementary actions dovetail together into an effective whole.

As described in Chapter 2, similar ideas can and have been incorporated in multiagent systems. Organizational structuring has been exploited in a variety of application domains for multiagent systems, such as disaster response and sensor networks. Since the sensor network application has been a mainstay of organizational structuring research for decades, we use it for illustrative purposes in this section.

The most obvious roles for agents in a distributed sensor network correspond with geographical regions: different sensor agents will be responsible for monitoring events near where they are located (or where they are now tasked with relocating to). Where sensor coverages overlap, responsibility for the overlapping region should be assigned, though perhaps not exclusively. That is, just as in human organizations where overlap between roles allows whomever is least burdened in the current situation to take on more of the shared responsibility, role overlap in multiagent systems also enables some degree of dynamic load balancing, and even fault tolerance.

Other forms of task decomposition within the distributed sensor network domain can lead to further refinement of roles. An agent with access to a particular sensory apparatus (e.g., acoustic instead of visual) might be given greater responsibility for monitoring for particular events. Agents might balance computational load by assigning responsibility for different kinds of phenomena among themselves. Some agents might be given greater responsibility for integrating interpretations from others rather than forming interpretations from raw data themselves.

### 3.2.1 Organizational Design

While the preceding says something about what an organizational structure does, the question remains about where it comes from. In general, the space of possible organizational designs for a non-trivial multiagent (including human) enterprise is vast, and the ability to predict organizational performance (particularly in human settings) is limited. Hence, while computational techniques have been used to study and extend organizational theory [10], no consensus strategy for forming organizations for systems of computational agents has emerged.

As outlined in Chapter 2, organizational designs can arise from the bottom up, by adopting and codifying emergent patterns of interactions between agents,

or can be constructed from the top down, by tasking one or more organizational designers with forming an organizational structure that the collection of agents then adopts. Both cases involve a search over (part of) the space of designs.

To make the design process more concrete, we here summarize one approach developed by Sims, Corkill, and Lesser [57]. A core idea is to view the design of an organization much like the creation of a hierarchical plan: given goals and environmental conditions, decompose the goals into component subgoals, identify agent roles whose preconditions are met by the environment and whose expected effects match the subgoals, and compose an organization out of the resultant roles. Then, match agents to the roles to instantiate the organization.

More precisely, the ORGANIZATIONSEARCH algorithm takes a sorted list of candidate partial organizations, and steps through the list until the following procedure returns:

1. Generate expansions of the candidate partial organization by finding a goal leaf in the decomposition hierarchy that has not been fully bound, and replacing it with either a role-goal binding indicating how it could be achieved, or with a subgoal tree that further decomposes it into subgoals.

2. Repeat step 1 until all leaves have associated roles.

3. Then use information about agent capabilities to assign agents to the roles.

4. If all roles can be assigned, return the organizational design, else return failure, triggering the search to continue with other candidate decompositions.

The design search begins with the candidate partial plan corresponding to the organization's top-level goal(s), and terminates as soon as a completely instantiated organizational design has been found. Because the candidate list is kept sorted using a heuristic, the first successful returned design is adopted; even though a better design might be possible, the costs of an exhaustive search for it argues for heuristic termination.

The above algorithmic outline ignores a variety of details about how knowledge about decompositions and agent capabilities are collected, stored, and retrieved, and about complications that arise from, for example, assigning a single role to multiple agents that then themselves need to coordinate. Yet, it is fundamentally like a planning algorithm. As has been pointed out elsewhere, the distinction between an organizational role and an abstract plan step is blurry [21]. In both cases, the agent is expected to dynamically elaborate the specification given its current circumstances, with the expectation that any suitable elaboration will fulfill the responsibilities to the rest of the organization/plan.

### 3.2.2 Organizational Execution and Functionally-Accurate Cooperation

Agents working in a distributed sensor network lack global awareness of the problem, and thus cooperate by forming local interpretations based on their local sensor data and the tentative partial interpretations received from others, and then sharing their own tentative partial interpretations. As a result, these agents need to cooperate to solve their subtasks, and might formulate tentative results along the way that turn out to be unnecessary. This style of collective problem solving has been termed functionally accurate (it gets the answer eventually, but with possibly many false starts) and cooperative (it requires iterative exchange) [40].

Functionally-accurate cooperation has been used extensively in distributed problem solving for tasks such as interpretation and design, where agents only discover the details of how their subproblem results interrelate through tentative formulation and iterative exchange. For this method to work well, participating agents need to treat the partial results they have formulated and received as tentative, and therefore might have to entertain and contrast several competing partial hypotheses at once. A variety of agent architectures can support this need; in particular, blackboard architectures [15] have often been employed as semi-structured repositories for storing multiple competing hypotheses.

Exchanging tentative partial solutions can impact completeness, precision, and confidence. When agents can synthesize partial solutions into larger (possibly still partial) solutions, more of the overall problem is covered by the solution. When an agent uses a result from another to refine its own solutions, precision is increased. And when an agent combines confidence measures of two (corroborating or competing) partial solutions, the confidence it has in the solutions changes. In general, most distributed problem-solving systems assume similar representations of partial solutions (and their certainty measures), which makes combining them straightforward, although some researchers have considered challenges in crossing between representations, such as combining different uncertainty measurements [68].

In functionally-accurate cooperation, the iterative exchange of partial results is expected to lead, eventually, to some agent having enough information to keep moving the overall problem solving forward. Given enough information exchange, therefore, the overall problem will be solved. Of course, without being tempered by some control decisions, this style of cooperative problem solving could incur dramatic amounts of communication overhead and wasted computation. For example, if agents share too many results, a phenomenon called *distraction* can arise: it turns out that they can begin to all gravitate toward doing the same problem-solving actions (synthesizing the same partial results into more complete solutions). That is, they all begin exploring the same part of the search space. For this reason, limiting communication is usually a good idea, as is giving

agents some degree of skepticism in how they assimilate and react to information from others. We address these issues next.

Organizational structuring can provide the basis for making good decisions about where agents should direct their attention and apply their communication resources. The organization defines control and communication protocols between agents by providing messaging templates and patterns to agents that trigger appropriate information exchange. As a simple example, the organization can provide an agent with simple communication rules, such that if the agent creates a local hypothesis that matches the rule pattern (e.g., characterizes an event near a boundary with other agents), then the agent should send that hypothesis to the specified agents. Similarly, if an agent receives a hypothesis from another, the organizational structure can dictate the degree to which it should believe and act on (versus being skeptical about) the hypothesis.

Organization structures thus provide static guidelines about who is generally interested in what results. But this ignores timing issues. When deciding whether to send a result, an agent really wants to know whether the potential recipient is likely to be interested in the result now (or soon). Sending a result that is potentially useful but that turns out not to be at best clutters up the memory of the recipient, and at worst can distract the recipient away from the useful work that it otherwise would have done. On the other hand, refraining from sending a result for fear of these negative consequences can lead to delays in the pursuit of worthwhile results and even to the failure of the system to converge on reasonable solutions at all because some links in the solution chain were broken.

When cluttering memory is not terrible and when distracting garden paths are short, then the communication strategy can simply be to send all partial results. On the other hand, when it is likely that an exchange of a partial result will distract a subset of agents into redundant exploration of a part of the solution space, it is better to refrain, and only send a partial result when the agent that generated it has completed everything that it can do with it locally. For example, in a distributed theorem-proving problem, an agent might work forward through a number of resolutions toward the sentence to prove, and might transmit the final resolvent that it has formed when it could progress no further.

Between the extremes of sending everything and sending only locally-complete results are a variety of gradations [22], including sending a small partial result early on (to potentially spur the recipient into pursuing useful related results earlier). For example, in a distributed vehicle monitoring problem, sensing agents in neighboring regions need their maps to agree on how vehicles move from one region to the other. Rather than waiting until it forms its own local map before telling its neighbor, an agent can send a preliminary piece of its map near the boundary early on, to stimulate its neighbor into forming a complementary map

(or determining that no such map is possible and that the first agent is pursuing a dubious interpretation path).

So far, we have concentrated on how agents decide when and with whom to voluntarily share results. But the decision could clearly be reversed: agents could only send results when requested. When the space of results formed is large and only a few are really needed by others, then sending requests (or more generally, goals) to others makes more sense. This strategy has been explored in distributed vehicle monitoring [17], as well as in distributed theorem proving [25, 42].

It is also important to consider the delays in iterative exchange compared to a blind inundation of information. A request followed by a reply incurs two communication delays, compared to the voluntary sharing of an unrequested result. But sharing too many unrequested results can introduce substantial overhead. Clearly, there is a trade-off between reducing information exchanged by iterative messaging versus reducing delay in having the needed information reach its destination by sending many messages at the same time. Sen, for example, has looked at this in the context of distributed meeting scheduling [52]. Our experience as human meeting schedulers tells us that finding a meeting time could involve a series of proposals of specific times until one is acceptable, or it could involve having the participants send all of their available times at the outset. Most typically, however, practical considerations leave us somewhere between these extremes, sending several well-chosen options at each iteration.

Finally, the communication strategies outlined have assumed that messages are assured of getting through. If messages get lost, then results (or requests for results) will not get through. But since agents do not necessarily expect messages from each other, a potential recipient will be unable to determine whether or not messages have been lost. One solution to this is to require that messages be acknowledged, and that an agent sending a message will periodically repeat the message (sometimes called "murmuring") until it gets an acknowledgment [41]. Or, a less obtrusive but more uncertain method is for the sending agent to predict how the message will affect the recipient, and to assume the message made it through when the predicted change of behavior is observed.

## 3.3 The Contract-Net Protocol and Role Assignment

The third and final category of techniques we discuss where coordination is done prior to making local planning and control decisions is the use of predefined protocols. An entire chapter of this book is dedicated to communication and protocols, and so we will not go into all of the gory details. But the important point from the perspective of this chapter is that protocols typically amount to predefined multiagent plan templates. These plan templates are intended to bring about a new joint state of the world, where the new state can include agents now knowing

things that they previously did not know, forging relationships and dependencies that mutually benefit them, making commitments that allow each to pursue some tasks with confidence that others will pursue related tasks, etc.

Formulating protocols is thus akin to formulating social laws or organizational structures: given properties of desired states of the world, predefine patterns of actions that if jointly followed will bring them about. In fact, the creation of choreographed service-oriented computing frameworks can treat the problem of composing and sequencing services as a planning problem (Chapter 3).

For the remainder of this section, however, we focus not on where protocols come from, but rather on how they can serve to control the interactions of agents toward a particular outcome. We will illustrate the ideas by examining one of the very first multiagent protocols, the contract-net protocol, and one of its first applications, which is to establish a distributed sensor network [18]. In distributed sensor network establishment (DSNE), roles (areas of sensing responsibility, responsibilities for integrating partial interpretations into more complete ones) need to be assigned to agents, where the population of agents might be initially unknown or dynamically changing. Thus, the purpose of the protocol is to exchange information in a structured way to converge on assignments of roles to particular agents.

At the outset, it is assumed that a particular agent is given the task of monitoring a wide geographic area. This agent has expertise in how to perform the overall task, but is incapable of sensing all of the area from its own locality. Therefore, the first step is that an agent recognizes that to perform its task better (or at all) it should enlist the help of other agents. As a consequence, it then needs to create subtasks to offload to other agents. In the DSNE problem, it can use its representation of the structure of the task to identify that it needs sensing done (and sensed data returned) from remote areas. Given this decomposition, it then uses the protocol to match these sensing subtasks with available agents.

The agent announces a request for bids for subtask. The important aspects of the announcement for our purposes here are the eligibility specification, the task abstraction, and the bid specification. (Attributes of message structures are described more fully in Chapter 3.) To be eligible for this task requires that the bidding agent have a sensor position within the required sensing area and that it have the desired sensing capabilities. Agents that meet these requirements can then analyze the task abstraction (what, at an abstract level, is the task being asked of the bidders?) and can determine the degree to which it is willing and able to perform the task. An eligible agent can then bid on the task, where the content of a bid is dictated by the bid specification.

The agent with the task receives back zero or more bids. If it gets no bids, then it can give up, try again (since the population of agents might be changing),

broaden the eligibility requirements to increase the pool of potential bidders, or decompose the task differently to target a different pool of bidders. Even if it gets back bids, it could be that none are acceptable to it, and it is as if it got none back. If one or more is acceptable, then it can award the sensing subtask to one (or possibly several) of the bidding agents. Note that, because the agent with the task has a choice over what it announces and what bids it accepts, and an eligible agent has a choice over whether it wants to bid and what content to put into its bid, no agent is forced to be part of a contract. The agents engage in a rudimentary form of negotiation, and form teams through *mutual selection*.

# 4 Local Planning Prior to Coordination

In some problem domains, predicting and pre-arranging the resolution of all possible interactions can be difficult and costly. For example, consider an environment where agents might pursue a wide variety of goals largely independently, but where aspects of the environment are shared such that how one agent affects the environment can impact how (and even whether) another agent can achieve its goals. Anticipating and planning for every possible interaction might be overkill. Instead, coordination should depend on the actual plans, and hence emergent interactions, of the agents in the current circumstances.

This viewpoint is appealing from the perspective of "divide and conquer" problem solving. The notion is to divide the problem up such that agents initially treat their own local problems as being independent, and thus each agent can formulate its own plan concurrently with the planning of other agents. After formulating their separate plans, then, the agents need to coordinate their plans to resolve their unintended interactions. We will refer to the problem of resolving interactions between separately-formed agent plans as the *multiagent plan coordination problem (MPCP)*.

As has been noted by a variety of people (dating back to Conry et al. [13], and recently by Nassim et al. [46]), this view of multiagent planning is compatible with a distributed constraint satisfaction formulation, where the variables are the agents' plans, and the constraints enforce that the plans dovetail together suitably. In distributed constraint satisfaction approaches [67], each agent is responsible for some set of variables, where each variable has an associated (typically finite) domain of values, and whose value assignment can be constrained depending on the assignments of other variables (possibly belonging to other agents). Traditionally, distributed constraint satisfaction algorithms involve asynchronous exchanges of tentative value assignments to (some of) the variables, and then information about constraint violations that trigger one or more agents to revise their tentative assignments. Parallel search can speed up finding a satisfying assignment, but care

must be taken to ensure the process is complete and terminates.

The MPCP differs from typical distributed constraint satisfaction problems in several important ways. First, the domain of possible "values" for an agent's plan "variable" is usually large (even infinite), and expensive to construct. Hence, there is a desire to generate as few elements of the variable domains as possible before converging on a joint plan. Second, the "constraints" between agents' variables are complex. It is non-trivial to assess whether the plans of two agents are compatible and will lead to some desired outcome state (or avoid an undesired state) if executed asynchronously.

Hence, the MPCP is generally solved in a sequential manner, possibly with backtracking, without assurances of finding an optimal joint plan. The basic outline of the algorithm is:

1. Each agent builds its local plan as if it were alone in the environment.

2. Agents directly, or through a more centralized mediary, identify potential problems that can arise during joint execution.

3. For problems that can arise, agents inject additional constraints into their plans (typically, over the timing of their actions relative to each other) to prevent such problems.

4. If all problems are prevented, then the agents are done. Otherwise, if some problems cannot be prevented, then one or more agents develops an alternative local plan, and the coordination problem repeats with the new portfolio of agent plans.

## 4.1 State-Space Techniques

The preceding algorithm sketch obviously leaves underspecified exactly how agents identify and rectify potential problems. One approach for doing so, reminiscent of GRAPHPLAN concepts [50, section 11.4], is to forward simulate the execution of agents' plans, step by step, and to detect whether an inconsistent state of the world arises, such as where a condition is simultaneously both established and undone, or when a single resource is assumed to be claimed by more than one agent at the same time. When such a state arises, the combination of actions that led to the inconsistency is analyzed, and one or more of those actions are prohibited to prevent the inconsistency. In essence, the agent(s) performing those action(s) are required to pause in the execution of their plan(s) at this step. Then, from the new resulting consistent state, again all agents (including those who paused) propose their next actions, and the process continues. If one or more agents is blocked from ever completing its plan, then the process can backtrack to

either pause different combinations of actions, or even to request some agents to form different local plans. It is also possible for some agent actions to be entirely skipped over, if serendipitously some other agents had established the conditions that those actions were intended to establish.

Ephrati and Rosenschein [23, 24] have formulated a more robust version of this approach. Their *plan combination search* approach to multiagent plan coordination begins with each agent constructing a set of possible local plans, rather than just one specific local plan which might be incompatible with local plans of others. During the search, the agents' sets of plans gets refined to converge on a nearly optimal combination. The search process avoids commitment to sequences of actions by specifying sets of *propositions* that hold as a result of action sequences instead of fully grounded states. The agents search by proposing, given a particular set of propositions about the world, the changes to that set that they each can make with a single action from their plans. These are all considered so as to generate candidate next sets of propositions about the world, and these candidates are ranked using an A* heuristic. Specifically, the cost of reaching the candidate set of propositions is summed with the total of each agent's estimated cost to achieve its goal based on its plans. The best candidate is chosen and the process repeats, until no agent wants to propose any changes (each has accomplished its goal).

Ephrati and Rosenschein illustrate this approach using a simple problem of agents cooperatively constructing an arch by separately planning the construction of each upright as well as the lintel [23]. Note that, depending on the more global movement of the plan, an agent will be narrowing down the plan it expects to use to accomplish its own private goals. Thus, agents are simultaneously searching for which local plan to use as well as for synchronization constraints on their actions, since in many cases the optimal step forward in the set of achieved propositions might omit the possible contributions of an agent, meaning that the agent should not perform an action at that time.

## 4.2 Plan-Space Techniques

The plan-space formulation of the MPCP below follows from Russell and Norvig's presentation of partial-order planning [50]. We begin with a brief refresher on single-agent partial-order planning, and then provide definitions and algorithms that extend it to the multiagent case.

### 4.2.1 Single-Agent Plans

A single-agent plan is a total or partial ordering of steps that will advance an agent from its initial state $I$ to a state that satisfies its goal conditions $G$. A plan

step is a fully grounded (or variable free) instance of an operator from the agent's set of operators. An operator $a$ in this representation has a set of preconditions ($pre(a)$) and postconditions ($post(a)$), where each condition $c \in pre(a) \cup post(a)$ is a positive or negative (negated) first-order literal. The set $pre(a)$ represents the set of preconditions that must hold for the agent to carry out operator $a$, and the set $post(a)$ represents the postconditions, or effects, of executing the operator on an agent's world state.

A standard formulation of a single-agent plan is a partial-order, causal-link (POCL) plan. POCL plans capture temporal and causal relations between steps in the partial-order plan. The definition of a POCL plan here is based on Bäckström [2], though it follows common conventions in the POCL planning community [63] to include special steps representing the initial and goal states of the plan.

**Definition 11.1** *A **POCL plan** is a tuple $P = \langle S, \prec_T, \prec_C \rangle$ where $S$ is a set of plan steps (operator instances), $\prec_T$ and $\prec_C$ are (respectively) the temporal and causal partial orders on $S$, where $e \in \prec_T$ is a tuple $\langle s_i, s_j \rangle$ with $s_i, s_j \in S$, and $e \in \prec_C$ is a tuple $\langle s_i, s_j, c \rangle$ with $s_i, s_j \in S$ and where $c$ is a condition. A POCL plan models the agent's initial state using an* init *step, init $\in S$, and the agent's goal using a* goal *step, goal $\in S$, where $post(init) = I$ (the initial state conditions), and $pre(goal) = G$ (the goal conditions).*

Elements of $\prec_T$ are commonly called **ordering constraints** on the steps in the plan. A partial-order plan has the following properties:

- $\prec_T$ is *irreflexive* (if $s_i \prec_T s_j$ then $s_j \not\prec_T s_i$).
- $\prec_T$ is *transitive* (if $s_i \prec_T s_j$ and $s_j \prec_T s_k$ then $s_i \prec_T s_k$).

Elements of $\prec_C$ are the causal links, representing causal relations between steps, where causal link $\langle s_i, s_j, c \rangle$ represents the fact that step $s_i$ achieves condition $c$ for step $s_j$. The presence of a causal link in a plan implies the presence of an ordering constraint.

The *single-agent planning problem* can be seen as the problem of transforming an inconsistent POCL plan into a consistent POCL plan. This is done by searching through the space of possible POCL plans, identifying the consistency flaws in the current POCL plan under consideration, and iteratively repairing them to produce a state representing a consistent POCL plan. Flaws include causal-link conflicts and open preconditions. The presence of a causal-link conflict in a plan indicates that, for some causal link $\langle s_i, s_j, c \rangle$, there exist executions (linearizations) of the partial-order plan where a step $s_k \in S$ negates condition $c$ after it is produced by $s_i$ but before it can be utilized by step $s_j$. Given a conflict between a step $s_k$ and a causal link $\langle s_i, s_j, c \rangle$, the standard method to resolve it is to add either $\langle s_k, s_i \rangle$ or

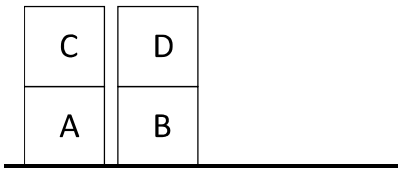


Figure 11.1: Initial state for simple blocks world problem.

$\langle s_j, s_k \rangle$ to $\prec_T$. That is, order the threatening step either before or after the link. An open precondition $c$ of a plan step $s_j \in S$ can be satisfied by adding a causal link $\langle s_i, s_j, c \rangle$ where step $s_i \in S$ establishes the needed condition (and $s_i$ is not ordered after $s_j$). If this requires that a new step $s_i$ is added to $S$, then its preconditions become new open preconditions in the plan.

As a simple example, consider the blocks world situation portrayed in Figure 11.1. Say Agent1 has a goal of achieving a state where block A is on block B. Using the POCL algorithm, it creates the plan shown in Figure 11.2, where actions and their parameters are given in the squares, their preconditions (postconditions) are given to the left (right) of the square, causal links are the narrow black arrows, and ordering constraints are the wide gray arrows. Notice that the plan is partially ordered, in that before block A can be stacked on B, both blocks must be cleared, but they can be cleared in either order. Finally, in Figure 11.3 is a plan for Agent2 to stack block B on C. Neither agent cares where block D ends up.

### 4.2.2 Multiagent Plans

To extend the notation and definitions of POCL plans to the multiagent case, we first address the issue of action concurrency. A single-agent usually takes only one action at a time, and thus a POCL plan such as in Figure 11.2 will be linearized before or during execution. If an agent can execute multiple actions in parallel, unordered actions that are eligible for execution are assumed to be executed concurrently. The POCL plan representation cannot express that *some* unordered steps to be executed in parallel but not others. To make this distinction, we extend from Bäckström [2] the idea of a *parallel* plan, to define a *parallel POCL plan*.

**Definition 11.2** *A **parallel POCL plan** is a tuple* $P = \langle S, \prec_T, \prec_C, \#, = \rangle$ *where* $\langle S, \prec_T, \prec_C \rangle$ *is the embedded POCL plan, and "#" and "=" are symmetric* non-concurrency *and* concurrency *relations over the steps in S, respectively.*

The relation $\langle s_i, s_j \rangle \in =$ means that $s_i$ and $s_j$ are required to be executed simultaneously. For example, if a plan has multiple goal steps and is intended to reach a state where all goals are satisfied simultaneously, then all pairs of goal steps would be elements of $=$. The relation $\langle s_i, s_j \rangle \in \#$ is equivalent to the statement

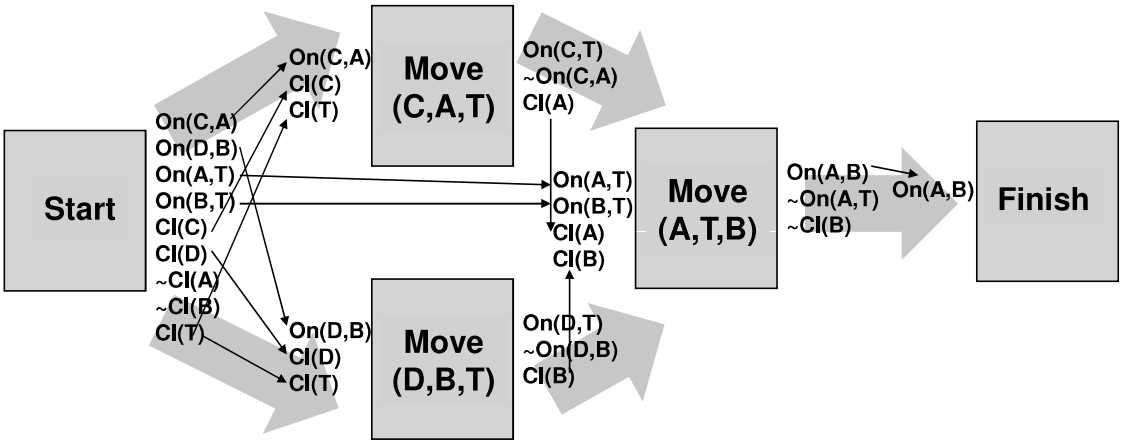


Figure 11.2: Single POCL plan for stacking block A on block B.

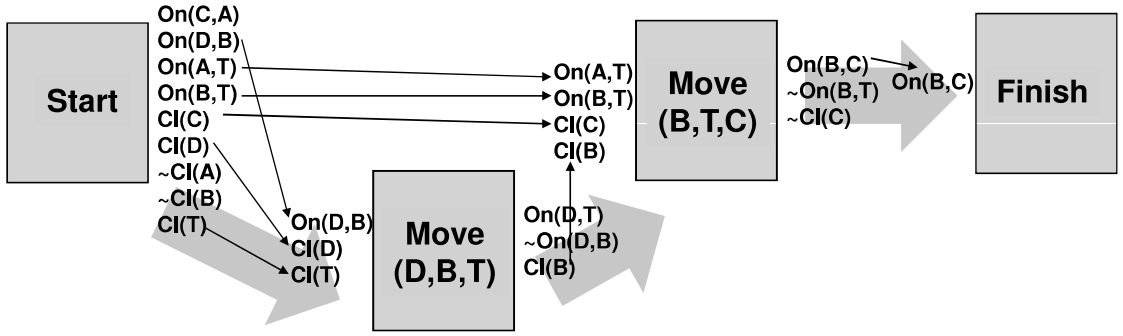


Figure 11.3: Single POCL plan for stacking block B on block C.

$(s_j \prec_T s_i) \vee (s_i \prec_T s_j)$. The # relation is needed because we make the assumption that parallel plans obey the *post-exclusion principle* [2], which states that actions cannot take place simultaneously when their postconditions are not consistent. The # and $=$ are disjoint sets, as two steps cannot be required to be concurrent and non-concurrent.

Given this definition of a parallel plan, it is clear that a partial-order (POCL) plan $P$ is a specialization of a parallel (POCL) plan $P^*$ in which either all pairs of steps in $P^*$ are in # (if it is assumed an agent can do only one action at a time) or none of them are (if all unordered actions are assumed to be concurrently executable). Likewise, a POCL plan implicitly requires that $=$ be empty (unless a

single agent can execute multiple steps concurrently).

Because of the post-exclusion principle, parallel plans have an additional source of plan flaws:

**Definition 11.3** *A **parallel-step conflict** exists in a parallel plan when there are steps $s_j$ and $s_i$ where post$(s_i)$ is inconsistent with post$(s_j)$, $s_j \not\prec_T s_i$, $s_i \not\prec s_j$ and $\langle s_i, s_j \rangle \notin \#$.*

However, unlike open conditions and causal-link conflicts, parallel-step conflicts can always be resolved, no matter what other flaw resolution choices are made. Recall that to repair a parallel-step conflict between steps $s_i$ and $s_j$, we need only ensure that the steps are non-concurrent, either by adding $s_i \prec_T s_j$ or $s_j \prec_T s_i$ to the plan. Given an acyclic plan $P$, there will always be at least one way of ordering every pair of steps in the plan such that the plan $P$ remains acyclic. This can be trivially shown by considering the four possible existing orderings of any pair of steps $s_i$ and $s_j$ in plan $P$. First, $s_i$ and $s_j$ could be unordered. In this case, we can add either $s_i \prec_T s_j$ or $s_j \prec_T s_i$ to the plan without introducing cycles in the network of steps. Second, $s_i \prec_T s_j$ is in the plan, in which case the parallel-step conflict has already been resolved. The same is true when $s_j \prec_T s_i$ is in the plan. Finally, $s_i \prec_T s_j$ *and* $s_j \prec_T s_i$ could both hold in the plan, but in this case the plan already has a cycle, and so repairing the parallel-step conflict becomes moot.

The parallel plan model captures the idea of concurrency, but it is not rich enough to describe the characteristics of a multiagent plan, in which we also need to represent the agents involved, and to which actions they are assigned. To do so, we extend the definition of a parallel plan to a multiagent parallel POCL plan.

**Definition 11.4** *A **multiagent parallel POCL plan** is a tuple $M = \langle A, S, \prec_T, \prec_C, \#, =, X \rangle$ where $\langle S, \prec_T, \prec_C, \#, = \rangle$ is the embedded parallel POCL plan, A is the set of agents, and X is a set of tuples of form $\langle s, a \rangle$, representing that the agent $a \in A$ is* assigned *to execute step s. A multiagent plan models the agents' initial states using* init *steps, $init_i \in S$, and the goals of the agents using a set of goal steps, $goal_i \in S$, where the preconditions of the* goal *steps represent the conjunctive goal that the plan achieves, and the postconditions of the* init *steps represent features of the agents' initial states before any of them take any actions.*

Figure 11.4 shows the (inconsistent!) multiagent parallel POCL plan composed of the two agents' individual plans from Figures 11.2 and 11.3. The dashed rectangles around the *init* steps and *goal* steps indicate that these pairs of steps are in the $=$ relation. That is, all *init* steps happen simultaneously (there is a single initial state whose conditions are the union of the *init* step postconditions), and all *goal* steps happen simultaneously (there is a final goal state for the multiagent system whose conditions are the union of the *goal* step preconditions).

Now, just as in the single-agent case, the *multiagent planning problem* can be solved by making an inconsistent multiagent parallel POCL plan consistent, where each plan step is assigned to an agent capable of executing the step. More formally, the multiagent plan coordination problem is the problem, given a set of agents $A$ and the set of their associated POCL plans $\mathbf{P}$, of finding a consistent and optimal multiagent parallel POCL plan $M$ *composed entirely of steps drawn from* $\mathbf{P}$ (in which agents are only assigned to steps that originate from their own individual plans) that results in the establishment of all agents' goals, given the collective initial state of the agents. The MPCP can thus be seen as a restricted form of the more general multiagent planning problem in which new actions are not allowed to be added to any agent's plan.

This definition of the MPCP imposes a set of restrictions on the kinds of multiagent plan coordination problems that can be represented. Because an agent can only be assigned steps that originated in its individual plan, this definition does not model coordination problems where agents would have to reallocate their activities. Further, because only individually-planned steps are considered, the definition does not capture problems where additional action choices are available if agents work together; that is, an agent when planning individually will not consider an action that requires participation of one or more other agents. Finally, in keeping within the "classical" planning realm, the definition inherits its associated limitations, such as assuming a closed world with deterministic actions where the initial state is fully observable.

For any given multiagent parallel plan, there may be many possible *consistent* plans one could create by repairing the various plan flaws. However, not all *consistent* plans will be *optimal*. Based on the assumptions outlined previously concerning the nature of the multiagent plan coordination problem (namely, that the final plan must be assembled solely from the original agents' plans), an optimal multiagent plan will be one that minimizes the total cost of the multiagent plan:

**Definition 11.5** ***Total step cost*** *measures the cost of a multiagent parallel plan by the aggregate costs of the steps in the plan.*

This simple, global optimality definition is not the only one that could be used for the MPCP, but correlates to the most widely-adopted single-agent optimality criterion. Other relevant definitions include ones minimizing the time the agents take to execute their plans (exploiting parallelism), maximizing the load balance of the activities of the agents, or some weighted combination of various factors.

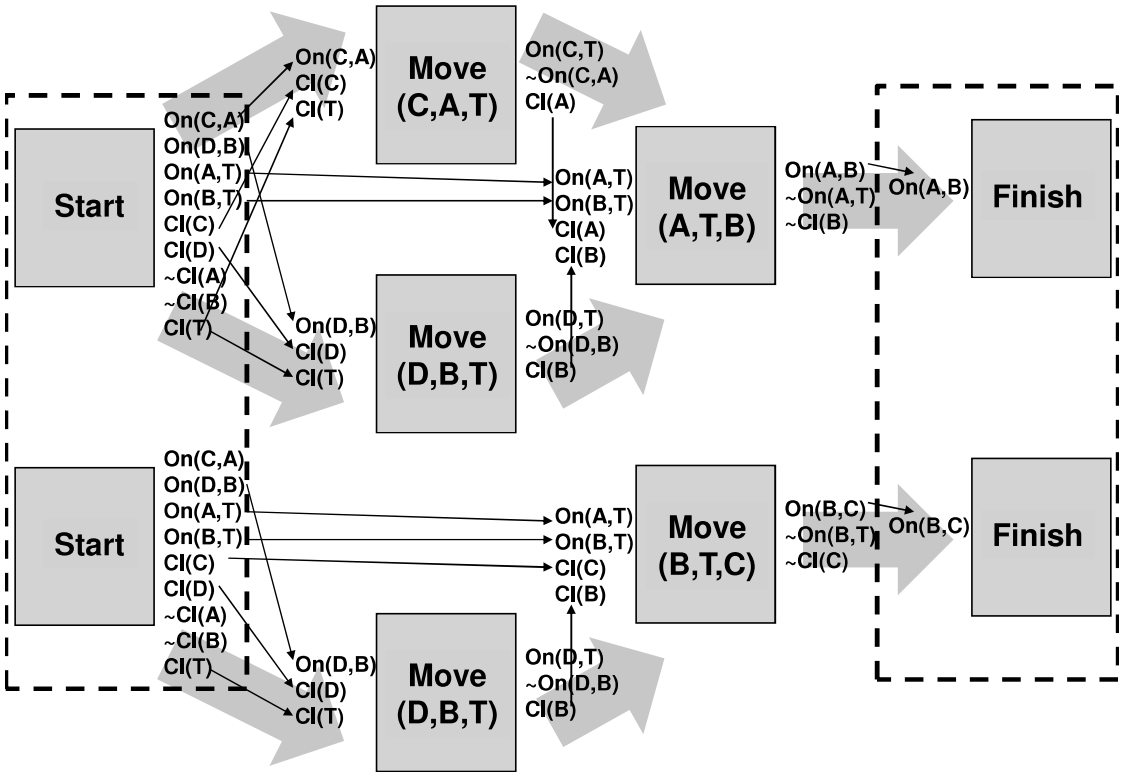


Figure 11.4: Initial (inconsistent) multiagent parallel POCL plan.

### 4.2.3 Multiagent Plan Coordination by Plan Modification

As illustrated in Figure 11.4, an initial multiagent parallel plan can simply be the union of the individual plan structures of the agents, and thus might contain flaws due to potential interactions between the individual plans. The initial multiagent plan can then be incrementally modified as needed (by both asserting new coordination decisions and retracting the individual planning decisions of the agents) to resolve the flaws. We call this approach coordination by *plan modification*.

From the initial (as yet uncoordinated) multiagent plan, plan coordination takes place by repairing any flaws due to interactions between the plans. The types of flaws are exactly the same as in parallel POCL plans: open preconditions, causal-link threats, and parallel-step flaws. Assuming each of the individual plans are consistent, there should be no open precondition flaws to resolve, at least to begin with. Causal link threats within each agent's plan should not exist, but new threats arise when an action in one agent's plan threatens a link in another

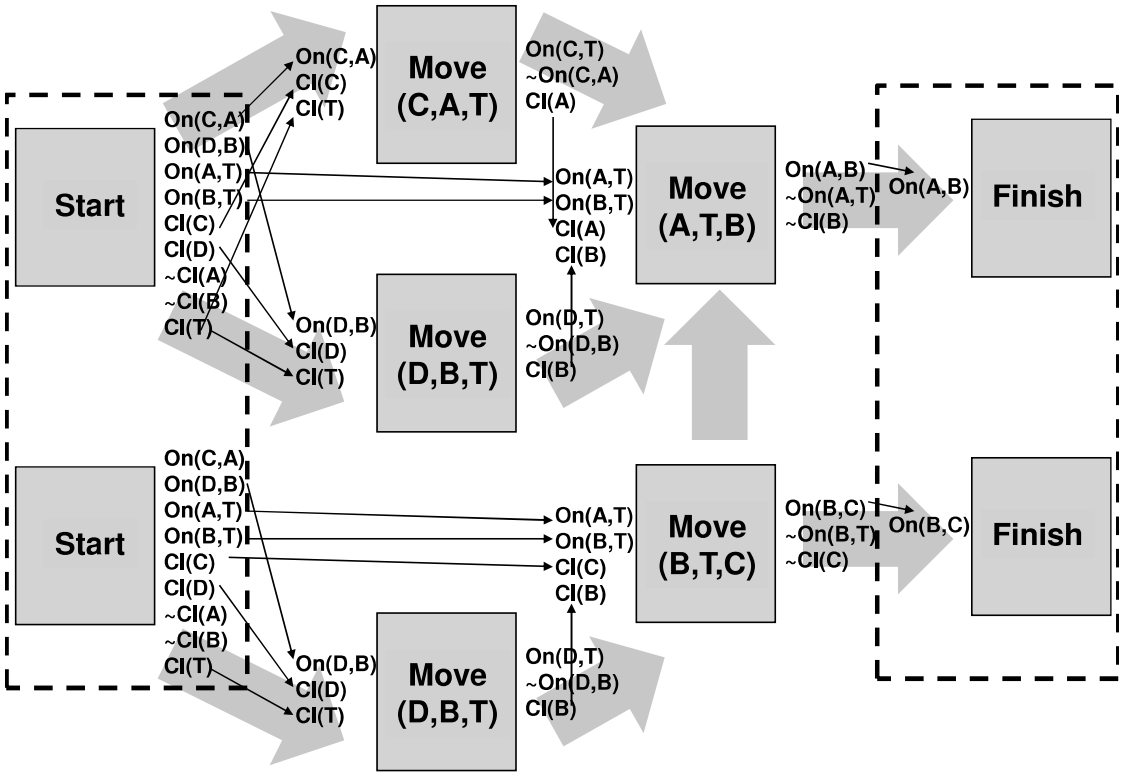


Figure 11.5: Ordering constraint added to resolve causal-link threat.

agent's plan. In the running example (Figure 11.4), Agent1's Move(A,T,B) step results in block B no longer being clear (¬Cl(B)), which threatens the causal link between Agent2's Move(D,B,T) and Move(B,T,C) steps. The flaw can be resolved by adding to the ordering constraints that Move(A,T,B) come after Move(B,T,C), as shown in Figure 11.5. Similarly, as before, parallel-step flaws can also be handled by adding in ordering constraints, though the running example has no such flaws.

The multiagent parallel POCL plan in Figure 11.5 could still be considered flawed, however, because Agent1 and Agent2 both are planning on moving block D from block B to the table. In the best case, one of these agents would execute the action, and then the other before attempting its action would recognize that it can simply skip the action because the desired effects are done. However, some plan execution systems would treat the situation as a deviation from expectations and attempt to repair the plan by inserting actions to (re)establish the conditions that the step expected. In other words, the second agent to execute the Move(D,B,T)

action might put block D back onto block B just so that it can move it off. This is obviously wasteful of time and energy. And, even worse, if the two agents were to attempt their Move(D,B,T) actions at about the same time, their effectors (grippers) might collide, and the agents might disable themselves!

The MPCP thus introduces a new type of flaw that affects the correctness, or at least the optimality, of the multiagent plan. Specifically, a step from some agent's plan could be *redundant* given the presence of steps in others' plans. Note that redundancy does not require that the agents seek the same effect. For example, if Agent1 had included action Move(D,B,T) to achieve On(D,T), while Agent2 had planned that action to achieve Cl(B), the action taken by one agent has the side effect of satisfying the other. In such cases, redundant steps may be able to be removed without introducing new open precondition flaws.

**Definition 11.6** *A plan step $s$ is **redundant** in a multiagent parallel POCL plan $M$ with steps $S$ when there exists a set of replacing steps $R$, where $R \subseteq S$, such that for each causal link of form $\langle s, s'', c \rangle$, it is also the case that $\exists s' \in R$ s.t. $c \in post(s')$.*

Redundancies can be discovered by altering the causal structure of the multiagent plan, by retracting some causal-link instantiation decisions and then asserting others to replace them (so as to prevent the introduction of open precondition flaws). To perform an adjustment of a single causal-link $l = \langle s_i, s_j, c \rangle$, we simply identify another step $s_k$ that also achieves condition $c$, and then change $l$ such that $l = \langle s_k, s_j, c \rangle$. If such an adjustment leaves $s_i$ with no outgoing causal links, then $s_i$ can be removed from the plan.

Note that the removal of a single redundant step may require many causal links to be adjusted as each outgoing causal link of a redundant step must be adjusted so that the redundant step is no longer causally necessary in the plan. Thus, a set of steps can collectively replace a single redundant step in a plan. Also note that removing plan steps provides an alternative way to resolve causal-link and parallel-step flaws: rather than adjust orderings for a step that threatens a link or introduces a potential inconsistency with another step, the step can be removed and the flaws go away. Finally, note that just because a redundancy exists, it does not mean that it can necessarily be exploited. As in a single-agent plan, the same action might need to be taken multiple times because its effects are necessarily undone by intervening actions. (The Towers of Hanoi puzzle is a familiar example of this.) During plan modification, if removal of such a step is attempted, the process of link adjustment would result in causal link threats that have no valid resolutions.

The plan modification algorithm (PMA) is shown in Algorithm 11.1. The PMA uses a best-first search in order to find the optimal solution. The search algorithm begins by initializing the search queue with the starting multiagent par-

```
Input  : an (inconsistent) multiagent parallel plan
Output: an optimal and consistent multiagent parallel plan or null plan
Initialize Solution to null;
Add input plan to search queue;
while queue not empty do
    Select and remove multiagent plan M from queue;
    if M not bounded by Solution then
        if (M passes Solution Test) and (steps in M < steps in Solution) then
            Solution = M;
        end
        Select and adjust a non-flagged causal-link in M;
        For each refinement, remove unnecessary steps in plan;
        Enqueue all plan refinements in search queue;
    end
end
repair parallel-step conflicts in Solution;
return Solution;
```

**Algorithm 11.1:** Multiagent plan coordination by plan modification.

allel plan, and by initializing the current best solution, *Solution*, to *null*. Then, while the queue is not empty, it selects a multiagent plan $M$ with the lowest total step cost from the queue.

A *bounding test* is applied to $M$ to determine whether it is possible for $M$ to have a lower total step cost than the best *Solution* found so far (if any). A lower bound on total step cost is computed for plan $M$ by working backwards from the goal steps to find all plan steps that contribute *flagged* causal links (as will shortly be explained) to achieving the goal. If the lower bound of $M$ is below the cost of *Solution*, the algorithm proceeds.

PMA next applies a *SolutionTest* to $M$, to derive a consistent solution from $M$, where any flaws in $M$ other than step redundancy flaws are iteratively resolved by adding ordering constraints. The *SolutionTest* thus conducts a depth-first search through the space of flaw resolutions to find a consistent solution, heuristically ordering the search to prioritize flaws for which there are fewer alternative resolutions, which is a minimum remaining values (MRV) heuristic [50]. If the consistent solution from $M$ has a lower total step cost than *Solution*, it replaces *Solution*.

The PMA algorithm then selects and adjusts a non-flagged causal link in $M$. In Figure 11.5, consider the causal link into condition Cl(B) for Agent1's Move(A,T,B) step. That causal link could originate from either Agent1's Move(D,T,B) step (as it does in the figure) or from Agent2's Move(D,T,B) step. The PMA makes copies of $M$ that differ only in this refinement of the source of

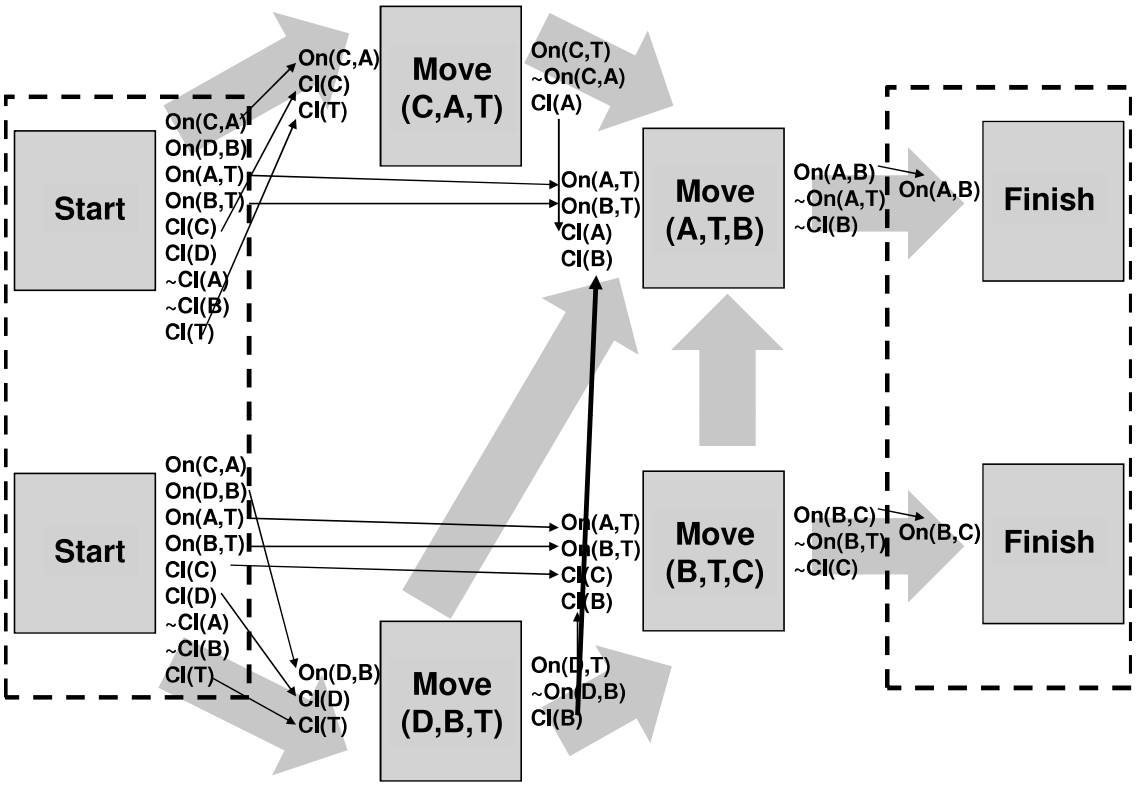


Figure 11.6: Solution multiagent parallel POCL plan.

the causal link, and in each "flags" the causal link so that it is not branched on again, because each of the refinements is added to the queue and might later be further modified. If in a refinement of *M* the redirection of causal links results in a plan step having no outgoing causal links, then that plan step is removed. When that refinement is enqueued, it will move forward in the queue since its total step cost will be lower.

The process of dequeuing, testing, and refining continues until the queue is empty, at which point the current value of *Solution* represents the lowest cost multiagent parallel plan derivable from the input plans. PMA then resolves any parallel-step conflicts in *Solution* (which, as was previously noted, must be resolvable), and returns *Solution*. In our simple example, the input from Figure 11.4 will lead to two possible solutions, one of which is shown in Figure 11.6 and the other where the Move(D,B,T) action is instead done by Agent1. These both have the same number of steps. If the total step cost function also considers parallel activity, the solution shown would be preferred.

## 4.3 Hierarchical Multiagent Plan Coordination

Hierarchical task network (HTN) planning is a method with algorithmic similarities to POCL planning. As per the summary in [50], single-agent HTN planning is a plan-space search method where the process focuses on *refining* abstract plans, rather than fixing flawed plans as in POCL planning. The idea is that the planning agent is endowed with a library of plans at various levels of abstraction, and the planning process involves refining a plan by iteratively replacing abstract steps with sequences of steps that are closer to being operational.

A familiar example of (human) HTN planning is planning a trip to attend a distant meeting. Given the distances involved, I might decide that my plan is to "fly" to the meeting. But I cannot execute such a plan without first refining it into steps of getting to a nearby airport, taking a flight to an airport near the meeting site, and then getting from that airport to the meeting. Each of these steps in turn needs to be refined further, until all the steps have been reduced to executable (primitive) actions. Given the same space of primitive actions, a state-space or POCL planner could in principle find the same plan, but the knowledge encoded in entries of the plan library (e.g., that flying involves going from where you are to an airport, taking a flight, and getting from the resulting airport to your final destination) can greatly streamline the planning process.

The same can be said for streamlining the multiagent planning process. The specification of hierarchical plans for teams of agents is part of a process that has been called *team-oriented programming* [49]. Such plans not only describe how high-level tasks are broken down into subtasks, but also how collaborative activities are broken down into different *roles*, similarly to relationships between agents in organizational structures (Section 3.2). Plan refinements associate with a subtask one or more roles that are responsible for that subtask, and ultimately the refinement process assigns an agent to each role. (Recovery from agent failure thus can be done by simply reassigning a role rather than replanning from scratch.) An agent can further refine an assigned subtask for its role using standard HTN planning. Timing and other relationships between different agents' tasks are specified in the team-oriented programming framework, but generic mechanisms for enforcing them can be automatically instantiated by a team-oriented programming infrastructure such as STEAM [59].

HTN planning representations have also been exploited as a means for flexibly solving multiagent plan coordination problems (Section 4). The insight is that, while an agent can only execute a plan at the level of primitive actions, multiagent coordination actions can (and often should) be based on more abstract levels of the hierarchy [11]. For example, two robots performing tasks in the rooms of a shared environment might avoid collisions by synchronizing to avoid ever occupying the same room, rather than incurring the expense of reasoning about colli-

1. Initialize the current abstraction level to the most abstract level.
2. Agents exchange descriptions of the plans and goals of interest at the current level.
3. Remove plans with no potential conflicts. If the set is empty, then done; otherwise determine whether to resolve conflicts at the current level or at a deeper level.
4. If conflicts are to be resolved at a deeper level, set the current level to the next deeper level and set the plans/goals of interest to the refinements of the plans with potential conflicts. Go to step 2.
5. If conflicts are to be resolved at this level:
   (a) Agents form a total order. Top agent is the current superior.
   (b) Current superior sends down its plan to the others.
   (c) Other agents change their plans to work properly with those of the current superior. Before confirming with the current superior, an agent also double-checks that its plan changes do not conflict with previous superiors.
   (d) Once no further changes are needed among the plans of the inferior agents, the current superior becomes a previous superior and the next agent in the total order becomes the superior. Return to step (b). If there is no next agent, then the protocol terminates and the agents have coordinated their plans.

**Algorithm 11.2:** Hierarchical behavior-space search.

sions at the primitive level. Thus, agents might communicate and coordinate at an abstract planning level. This not only can have computational benefits (fewer combinations of joint steps to reason about), but also can have flexibility benefits at execution time. For instance, in our example of robots in a shared workspace, if robots only coordinate at the level of entering and leaving rooms, then each robot retains flexibility to change its planned movements within a room without needing to renegotiate with the other. However, coordinating at an abstract level tends to lead to less efficient joint plans (e.g., a robot idling waiting for another to exit a room rather than carefully jointly working within the room). Further, to anticipate potential primitive interactions at abstract levels means that agents need to summarize for abstract steps the repercussions of the alternative refinements that might be made [11]. Algorithm 11.2 summarizes a simple algorithm for solving the multiagent plan coordination problem for agents that have hierarchical plans.

# 5 Decision-Theoretic Multiagent Planning

Decision-theoretic planning is aimed at choosing actions under uncertainty by maximizing the expected value of some performance measure called *utility*. Planning in this case explicitly factors the uncertainty about the outcomes of actions and the state of the domain, aiming to *optimize* utility rather than provably *satisfy* certain goals. For example, a decision-theoretic plan for a space exploration rover could maximize the *scientific return*, measured by the amount or the value of the collected data in a given mission, in the face of uncertainty about the pace of progress of the rover and amount of power left in its battery. When applied to a multiagent system, decision-theoretic planning optimizes simultaneously the local plan as well as coordination decisions. The value associated with each action is based on its impact on the domain, the information it transmits to other agents, and the information it obtains from the domain or other agents. Thus, a single planning process optimizes the comprehensive value, thereby optimizing both domain actions and coordination.

A standard framework to tackle planning under uncertainty is the *Markov decision process* (MDP) [47]. The model represents the domain using a set of states. It is designed for a *single* decision maker whose actions lead to stochastic transitions to new states and a reward that can depend on the action and outcome. The *partially observable MDP* (POMDP) is a generalization of the basic model that accounts for imperfect observations. In a POMDP, the decision maker receives *partial* information about the state of the world after taking each action. In that case, the agent can maintain a belief state (probability distribution over domain states), and must act without knowing the exact state of the world. One of the key observations that made it possible to solve single-agent POMDPs is that any POMDP can be viewed as a *belief-state* MDP – an MDP whose domain states are probability distributions over real-world states. Unfortunately, the same is not true in the multiagent case, making planning substantially more complicated. A range of exact and approximate dynamic programming algorithms have been developed for solving MDPs and POMDPs. These algorithms have been used in many practical applications.

A more general planning problem arises when two or more agents have to coordinate their actions. Imagine for example two space exploration rovers that conduct experiments as part of an overall mission. The value of the data collected by one rover may depend on the experiments performed by the other rover. Planning in this case becomes particularly complicated when each agent receives *different* observations and has different partial knowledge of the overall situation. We refer to such problems as *decentralized control problems*, indicating that all the agents control a single process in a collaborative manner, but must each act in a decentralized manner based on their own observations. Such decentralized multiagent

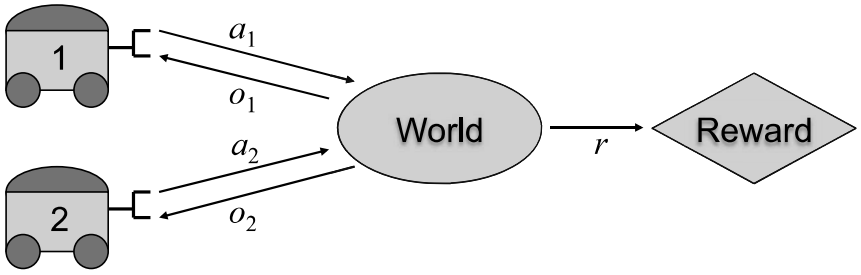


Figure 11.7: Illustration of a two-agent DEC-POMDP.

control problems are ubiquitous. Examples include coordination of mobile robots, load balancing for decentralized queues, target tracking in sensor networks, and monitoring of hazardous weather phenomena. This section describes models for representing such planning problems and algorithms for solving them.

## 5.1 Models for Decision-Theoretic Multiagent Planning

Natural extensions of MDPs and POMDPs to multiagent settings have been proposed and extensively studied since the late 1990s. We focus in this chapter on decentralized POMDPs (DEC-POMDPs) [7]. Figure 11.7 illustrates a DEC-POMDP with two agents. In each step, each agent takes an action, the joint set of actions causes a stochastic change in the state of the world, and a reward is generated based on the actions and their outcome. Then, each agent receives its own private observation and the cycle repeats.

**Definition 11.7 (DEC-POMDP)** *A decentralized partially observable Markov decision process (DEC-POMDP) is a tuple* $\langle I, S, \{A_i\}, P, \{\Omega_i\}, O, R, T \rangle$ *where*

- $I$ *is a finite set of agents indexed* $1, \ldots, n$.
- $S$ *is a finite set of states, with distinguished initial state* $s_0$ *or belief state* $b_0$.
- $A_i$ *is a finite set of actions available to agent* $i$ *and* $\vec{A} = \otimes_{i \in I} A_i$ *is the set of joint actions, where* $\vec{a} = \langle a_1, \ldots, a_n \rangle$ *denotes a joint action.*
- $P : S \times \vec{A} \to \Delta S$ *is a Markovian transition function.* $P(s'|s, \vec{a})$ *denotes the probability of a transition to state* $s'$ *after taking joint action* $\vec{a}$ *in state* $s$.
- $\Omega_i$ *is a finite set of observations available to agent* $i$ *and* $\vec{\Omega} = \otimes_{i \in I} \Omega_i$ *is the set of joint observations, where* $\vec{o} = \langle o_1, \ldots, o_n \rangle$ *denotes a joint observation.*
- $O : \vec{A} \times S \to \Delta\vec{\Omega}$ *is an observation function.* $O(\vec{o}|\vec{a}, s')$ *denotes the probability of observing joint observation* $\vec{o}$ *given that joint action* $\vec{a}$ *was taken and led to state* $s'$. *Here* $s' \in S$, $\vec{a} \in \vec{A}$, $\vec{o} \in \vec{\Omega}$.

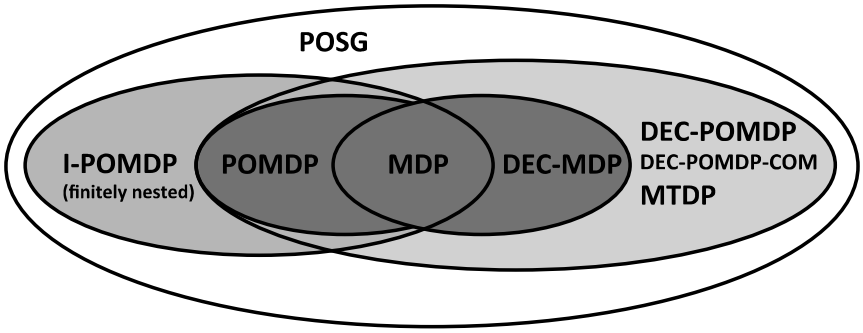


Figure 11.8: Relationship among several models for multiagent planning.

- $R : \vec{A} \times S \rightarrow \Re$ *is a reward function.* $R(\vec{a}, s')$ *denotes the reward obtained after joint action* $\vec{a}$ *was taken and a state transition to* $s'$ *occurred.*

The goal is to maximize the cumulative (discounted) reward over some finite horizon $T$ or over an infinite horizon. The model includes only one reward function, indicating that the agents operate collaboratively toward one objective. A special case of DEC-POMDP, called DEC-MDP, models situations in which the *combined* observations of all the agents provide perfect information about the underlying world state. DEC-MDPs extend *multiagent MDPs* (MMDPs), where each agent has full knowledge of the underlying world state [9]. Notice that a DEC-POMDP model is equivalent to a single-agent POMDP when $n = 1$.

Communication between agents can be modeled by a DEC-POMDP either implicitly or explicitly. Implicit communication occurs whenever one agent's actions affect another agent's observations. Explicit communication – exchanging messages between agents – can be represented by making the message a component of each observation. Each action in this case can be divided into two parts: a domain action that affects the state of the environment, and a communication action that affects the messages received by other agents.

A model equivalent to DEC-POMDP called *multiagent team decision problem* (MTDP) was introduced in 2002 [48]. DEC-POMDPs and MTDPs are a special case of *partially-observable stochastic games* (POSGs), which allow each agent to have a different objective encoded by a private reward function. Another related model that explicitly represents beliefs about other agents, called *interactive POMDP* (I-POMDP), was introduced in 2005 [28]. The relationships between the different models is illustrated in Figure 11.8, largely based on the analysis in [54].

To illustrate the problems and solution methods, we will use a simple toy problem called the *multiagent tiger problem*. This domain includes two agents, two states, three actions, and two observations, and was introduced by Nair et al. [44]. In this problem, the two agents are initially situated in a room with two

doors. Behind one door is a tiger and behind the other is a large treasure. Each agent may open one of the doors or listen. If either agent opens the door with the tiger behind it, a large penalty is given. If the door with the treasure behind it is opened and the tiger door is not, a reward is given. If both agents choose the same action (e.g., both open the same door), a larger positive reward or a smaller penalty is given to reward this cooperation. If an agent listens, a small penalty is given and an observation is seen that is a noisy indication of which door the tiger is behind. Once a door is opened, the game resumes from its initial state with the tiger, and treasure's locations randomly reshuffled.

This class of problems has raised several questions about the feasibility of decision-theoretic planning in multiagent settings: Are DEC-POMDPs significantly harder to solve than POMDPs? What features of the problem domain affect the complexity, and how? Is optimal dynamic programming possible? Can dynamic programming be made practical? Can locality of agent interaction be exploited to improve algorithm scalability? Research in recent years has significantly increased the understanding of these issues and produced a solid foundation for multiagent planning in stochastic environments. We describe below some of the key results and lessons learned.

### 5.1.1 Solution Representation and Evaluation

Solutions for decentralized control problems involve a set of *policies* – one per agent – that determine how each agent should act so as to maximize the overall reward. How can these policies be represented? In the case of finite-horizon problems, one can use *policy trees*. Each policy tree is a decision tree where each node is labeled with an action and branches are labeled with observations. Starting with the root node, at each step, each agent performs the action of the current node and then branches to a subtree based on the observation it receives. Sample optimal policy trees for the multiagent tiger problem with horizons 1–4 are shown in Figure 11.9. The actions are L (listen), OL (open left), and OR (open right). The observations are hl (hear tiger on left) and hr (hear tiger on right). Note that while the value generally grows with the horizon, there are some fluctuations when the added time allows for costly listening actions, but is not sufficient for establishing a reliable belief about the tiger's location.

Figure 11.10 shows a horizon 5 optimal solution (same tree for both agents). As this example illustrates, the size of policy trees grows exponentially with the horizon of the problem, making it hard to keep complete policy trees in large problems. And when the problem has an infinite horizon, policy trees can no longer be used to represent solutions. A common approach in that case is for each agent to summarize what it knows using finite memory and to represent policies using *finite-state controllers*. Each controller state represents an intermediate internal

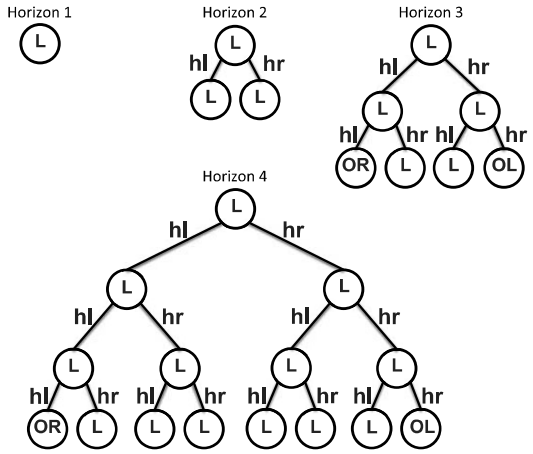


Figure 11.9: Optimal policy trees for the multiagent tiger problem with horizons 1–4. The policy trees of both agents are the same in this case. The expected values as a function of the horizon are: $V_1 = -2$, $V_2 = -4$, $V_3 = 5.19$, $V_4 = 4.80$.

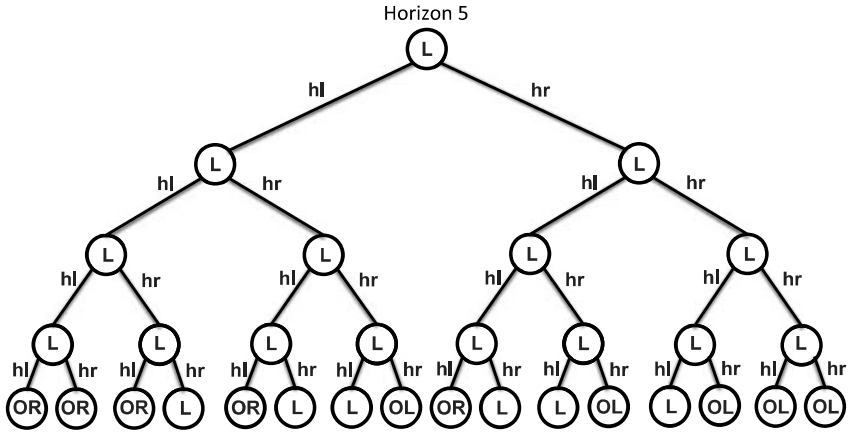


Figure 11.10: Optimal policy tree for the multiagent tiger problem with horizon 5. The expected value in this case is $V_5 = 7.03$.

memory state of the agent. Starting with an initial controller state, at each state an agent chooses an action based on its internal state and then branches to a new internal state based on the observation received. Both the action selection and controller transitions could be deterministic or stochastic; higher value could be obtained using stochastic mappings, but the optimization problem is harder.

Figure 11.11 shows optimal deterministic controllers for the infinite-horizon multiagent tiger problem with a discount factor of 0.9. The large arrow points to the initial state. The figure shows a one-node controller (same controller per

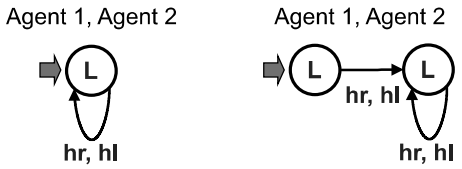


Figure 11.11: Optimal one-node and two-node deterministic controllers for the multiagent tiger problem.

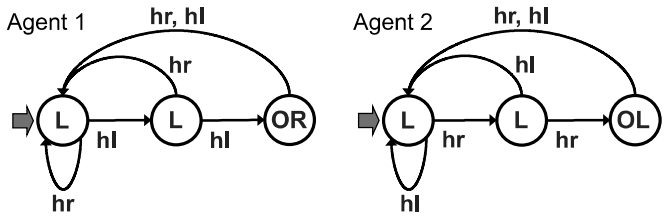


Figure 11.12: Optimal three-node deterministic controllers for multiagent tiger.

agent) with an expected value of $-20$, and a two-node controller, which represents the same policy and has the same value. With so little memory, the optimal policy is to listen all the time and not risk opening a door. Figure 11.12 shows an optimal deterministic solution with three-node controllers. The value in this case is $-14.12$ and the policies of the agents are different in this case. Figure 11.13 shows an optimal deterministic solution with four-node controllers. The value in this case is $-3.66$ and the policies of the agents are again different.

Figure 11.14 shows stochastic two-node controllers for this problem. The large arrow points to the initial state. Each state leads to multiple actions shown in rectangles with the probability of the action attached to the link. Each observation then leads to a stochastic transition to one of the two states. The value in this case is $-19.30$, a slight improvement over the two-node deterministic controller. A three-node stochastic controller (not shown) can achieve a value of $-9.94$.

Formally, these solutions assign a *local policy* to each agent $i$, $\delta_i$, which is a mapping from local histories of observations or internal memory states to actions. A *joint policy*, $\delta = \langle \delta_1, ..., \delta_n \rangle$, is a tuple of local policies, one for each agent.

For a finite-horizon problem with $T$ steps, the value of a joint policy $\delta$ with initial state $s_0$ is

$$V^{\delta}(s_0) = E\left[\sum_{t=0}^{T-1} R(\vec{a}_t, s_t) | s_0, \delta\right].$$

For an infinite-horizon problem, with initial state $s_0$ and discount factor $\gamma \in$

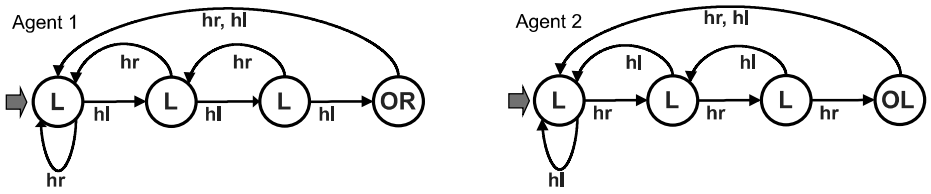


Figure 11.13: Optimal four-node deterministic controllers for multiagent tiger.

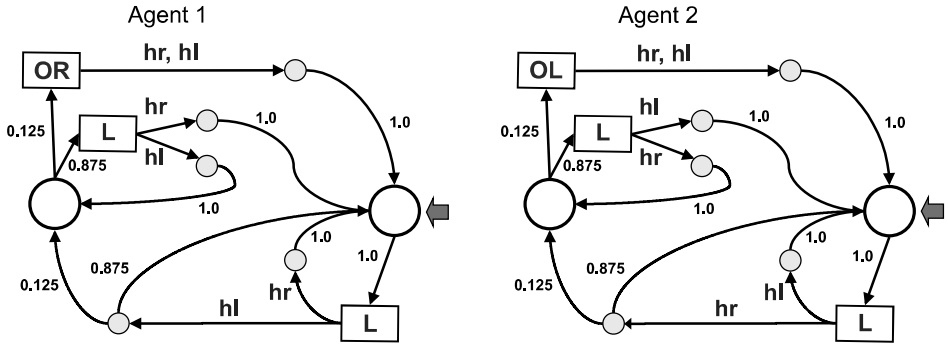


Figure 11.14: Stochastic two-node controllers for multiagent tiger.

$[0,1)$, the value of a joint policy $\delta$ is

$$V^{\delta}(s_0) = E\left[\sum_{t=0}^{\infty} \gamma^t R(\vec{a}_t, s_t) | s_0, \delta\right].$$

### 5.1.2 The Complexity of DEC-POMDPs

Complexity analysis of DEC-POMDPs has shown that the finite-horizon problem is NEXP-hard. What is striking is that the problem remains NEXP-hard even when restricted to a two-agent DEC-MDP. This is in contrast to the complexity of finite-horizon MDPs and POMDPs, which is P-complete and PSPACE-complete, respectively. Although it is not known whether the classes P, NP, and PSPACE are distinct, it is known that P $\neq$ NEXP, and thus DEC-POMDPs are provably intractable. Furthermore, assuming EXP $\neq$ NEXP, the problems take super-exponential time to solve in the worst case. These complexity results reveal some fundamental differences between centralized and decentralized control of Markov decision processes.

These results, however, represent the worst-case complexity of the general model. This presents the question of whether problems that arise in practice are intractable, and whether the structure and characteristics of some real-world problems make them fundamentally easier to solve. For example, mobile robots are

largely independent agents. They move and take actions using their private actuators, which are often totally independent of the actions of other robots operating in the environment. This property is called *transition independence*. Another useful property that is sometimes satisfied is *observation independence* – guaranteeing that the observations of one agent depend only on a component of the underlying state that is not affected by the actions of the other agents. Analysis of the problem shows that these assumptions could lead to a problem of lower complexity. For example, a DEC-MDP that satisfies transition and observation independence can be solved in exponential time [30].

## 5.2 Solving Finite-Horizon DEC-POMDPs

Dynamic programming algorithms for solving finite-horizon DEC-POMDPs construct and evaluate policy trees incrementally, starting with the final step and moving backward toward the first step. Policy trees of the current iteration become subtrees of the policies built in succeeding iterations. The key to success is reducing the amount of time and space either by performing pruning of irrelevant policies, or – in the case of approximate algorithms – pruning policies that are less likely to be useful.

Let $q_i$ denote a policy tree and $Q_i$ a set of policy trees for agent $i$. $Q_{-i}$ denotes the sets of policy trees for all agents but agent $i$. A joint policy $\vec{q} = \langle q_1, q_2, \cdots, q_n \rangle$ is a vector of policy trees and $\vec{Q} = \langle Q_1, Q_2, \cdots, Q_n \rangle$ denotes the sets of joint policies. Evaluating a joint policy $\vec{q}$ can be done as follows:

$$V(s, \vec{q}) = R(s, \vec{a}) + \sum_{s', \vec{o}} P(s'|s, \vec{a}) O(\vec{o}|s', \vec{a}) V(s', \vec{q}_{\vec{o}}) \tag{11.1}$$

where $\vec{a}$ are the actions at the roots of trees $\vec{q}$ and $\vec{q}_{\vec{o}}$ are the subtrees of $\vec{q}$ after obtaining observations $\vec{o}$.

In multiagent settings, agents have to reason about the possible future policies of the other agents in order to choose optimal actions. The standard *belief-state* of a POMDP – a probability distribution over world states – is no longer suitable. Instead, it is necessary to use a *multiagent belief state*, which is a probability distribution over system states and policies of all other agents: $b_i \in \Delta(S \times Q_{-i})$. Other forms of multiagent belief states could be used to capture the uncertainty about the beliefs or intentions of other agents, but the above form of belief state, also called *generalized belief state*, is the one used in this chapter because it is most commonly used in existing algorithms.

One of the early class of algorithms for solving DEC-POMDPs is the "Joint Equilibrium-Based Search for Policies" (JESP), which seeks to find a joint policy that is locally optimal. That is, the solution cannot be improved by any single

```
Generate a random joint policy
repeat
    foreach agent i do
        Fix the policies of all the agents except i
        for t=T downto 1 do // forwards
            Generate a set of all possible belief states:
            B^t(·|B^{t+1}, q^t_{-i}, a_i, o_i), ∀a_i ∈ A_i, ∀o_i ∈ Ω_i
        end
        for t=1 to T do // backwards
            foreach b^t ∈ B^t do
                Compute the best value for V^t(b^t, a_i)
            end
        end
        forall the possible observation sequences do
            for t=T downto 1 do // forwards
                Update the belief state b^t given q^t_{-i}
                Select the best action according to V^t(b^t, a_i)
            end
        end
    end
until no improvement in the policies of all agents
return the current joint policy
```

**Algorithm 11.3:** DP-JESP for DEC-POMDPs.

agent, *given* the policies assigned to the other agents. The best algorithm in this class, DP-JESP, incorporates three key ideas [44]. First, the policy of each agent is modified while keeping the policies of the others fixed. Second, dynamic programming is used to iteratively construct policies. Third, and most notably, only reachable belief states of the DEC-POMDP are considered for policy construction. This leads to a significant improvement, because there is only an exponential number of different belief states for one agent as opposed to the doubly exponential number of possible joint policies. Algorithm 11.3 summarizes the operation of DP-JESP [44]. This approach only guarantees local optimality and still leads to exponential complexity due to the exponential number of possible belief points. The algorithm could solve small benchmark problems up to horizon 7.

An *exact dynamic programming* (ExactDP) algorithm for solving DEC-POMDPs has been developed as well [32]. In every iteration, this algorithm first exhaustively *backups* the policy trees of the previous iteration, then prunes all the *dominated* policies. A policy of an agent is dominated by another policy when the value of *every* complete policy that includes it as a subtree can be improved (or

```
1 Initialize all depth-1 policy trees
2 for t=1 to T do // backwards
3 |   Perform full backup on Q⃗^t
4 |   Evaluate policies in Q⃗^{t+1}
5 |   Prune dominated policies in Q⃗^{t+1}
6 end
7 return the best joint policy in Q⃗^T for b^0
```

**Algorithm 11.4:** Exact DP for DEC-POMDPs.

preserved) by replacing it with the other policy. This property must be satisfied for *every* set of policies of the other agents and *every* belief state. It is clear that dominated policies are not needed to construct an optimal solution. Dominance can be tested efficiently using a linear program. The algorithm can solve partially-observable stochastic games with minimal changes, as it can produce *all* the non-dominated policies for each agent. This process, summarized in Algorithm 11.4, is the first DP algorithm that could produce a globally optimal solution of a DEC-POMDP. Unsurprisingly, this approach runs out of memory very quickly because the number of possible (non-dominated) joint policies grows doubly exponentially over the horizon. Even with very significant pruning, the algorithm can only solve small benchmark problems with horizons 4–5. But it introduces important pruning principles that prove useful in designing effective approximations.

Several improvements of the ExactDP algorithm have been proposed. Since some regions of the belief space are not reachable in many domains, the point-based DP (PBDP) algorithm computes policies only for the subset of reachable belief states [58]. Unlike DP-JESP, PBDP generates a full set of current-step policies and identifies the reachable beliefs by enumerating all possible top-down histories. This guarantees optimality with a somewhat more aggressive pruning. The worst-case complexity is thus doubly exponential due to the large number of possible policies and histories.

While the above algorithms introduced important ideas, it became clear that to improve scalability, it is necessary to perform more aggressive pruning and limit the amount of memory used by solution methods. The memory-bounded DP (MBDP) algorithm presented a new paradigm that allowed the algorithm to have linear time and space complexity with respect to the problem horizon [53]. MBDP, shown in Algorithm 11.5, employs top-down heuristics to identify the most useful belief states and keeps only a *fixed* number of policies selected based on these belief states. The number of policies maintained per agent is a constant called *maxTrees*. To assure linear space, however, it is not sufficient to limit the number of policies per agent, because the size of each policy tree grows expo-

Initialize all depth-1 policy trees
**for** $t$=*1* **to** $T$ **do** // backwards
  Perform full backup on $\vec{Q}^t$
  Evaluate policies in $\vec{Q}^{t+1}$
  **for** $k$=*1* **to** *maxTrees* **do**
    Select a heuristic $h$ from the heuristic portfolio
    Generate a state distribution $b \in \Delta(S)$ using $h$
    Select the best joint policy $\vec{q}^{\,t+1}$ in $\vec{Q}^{t+1}$ for $b$
  **end**
  Prune all the policies except the selected ones
**end**
**return** the best joint policy in $\vec{Q}^T$ for $b^0$

**Algorithm 11.5:** Memory-bounded DP for DEC-POMDPs.

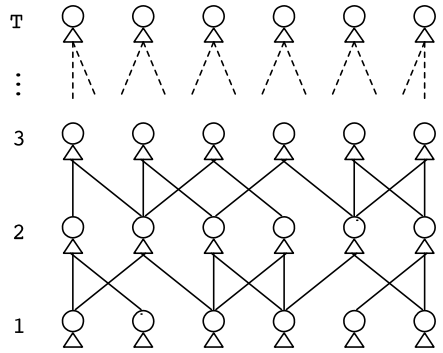


Figure 11.15: A set of *maxTrees* policy tree can be represented compactly by reusing a fixed number of *maxTrees* subpolicies of the previous level.

nentially with the horizon. To address that, MBDP deploys an efficient method to reuse subpolicies in a given policy tree. At each level, the new branches of the tree point to one of the *maxTrees* policies of the previous level, as illustrated in Figure 11.15. This memory-bounded policy representation enables the algorithm to solve much larger problems with essentially arbitrary horizons. A number of algorithmic improvements in MBDP and its variants have made it possible in recent years to solve effectively larger problems using dozens of *maxTrees* per level.

## 5.3 Solving Infinite-Horizon DEC-POMDPs

As illustrated earlier, finite-state controllers can be used to summarize unbounded observation histories using finite memory. One controller is used per agent, where

the state of the controller changes based on the observation sequence of the agent, and in turn the agent's actions are based on the state of its controller. When these functions are deterministic, optimizing controllers can be tackled using standard heuristic search methods. When these mappings are stochastic, the search for optimal controllers becomes a continuous optimization problem. We describe in this section two approaches for optimizing such stochastic controllers.

**Definition 11.8 (Local finite-state controller)** *Given a DEC-POMDP, a local finite-state controller for agent $i$ is a tuple $\langle Q_i, \psi_i, \eta_i \rangle$, where $Q_i$ is a finite set of controller nodes, $\psi_i : Q_i \to \Delta A_i$ is a stochastic action selection function, and $\eta_i : Q_i \times A_i \times O_i \to \Delta Q_i$ is a stochastic transition function.*

An *independent joint controller* is a set of local finite-state controllers, one for each agent, that together determine the conditional distribution $P(\vec{a}, \vec{q}' | \vec{q}, \vec{o})$. The controllers are *independent* in that the local memory state transitions and action selection functions of one agent are independent of the memory states and observations of the other agents, making the policy suitable for decentralized operation.

### 5.3.1 Correlated Joint Controllers

While agents do not have access to the local information of other agents in a DEC-POMDP, they *can* benefit from performing correlated actions. This can be achieved using a *correlation device* – a mechanism that can facilitate coordination using an additional finite-state controller whose state is accessible by all the agents [6]. The correlation device mimics a random process that is independent of the controlled system. Agents use the extra signal from the device to select actions, but they cannot control the correlation device. Such mechanism can be implemented in practice by giving each agent the same stream of random bits.

**Definition 11.9 (Correlation device)** *A correlation device is a tuple $\langle C, \psi \rangle$, where $C$ is a finite set of states, and $\psi : C \to \Delta C$ is a state transition function. At each time step, the device makes a transition and all the agents observe its new state.*

The definition of a local controller can be extended to consider the shared signal $c$ provided by a correlation device. The local controller for agent $i$ becomes a conditional distribution of the form $P(a_i, q_i' | c, q_i, o_i)$. A correlation device together with the local controllers for each agent form a joint conditional distribution $P(c', \vec{a}, \vec{q}' | c, \vec{q}, \vec{o})$, called a *correlated joint controller*.

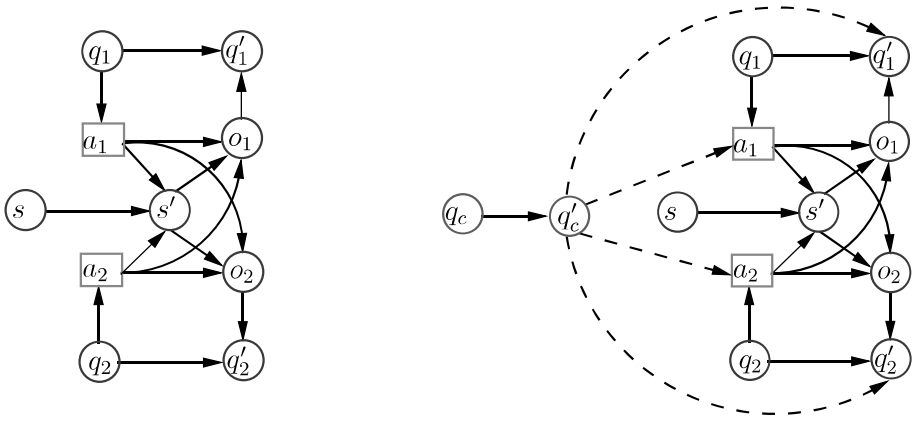


Figure 11.16: A slice of a two-agent DEC-POMDP where actions are selected based on internal states $q_i$ with (right) and without (left) a correlation device $q_c$.

The value of a correlated joint controller can then be computed by solving a set of linear equations, one for each $s \in S$, $\vec{q} \in \vec{Q}$, and $c \in C$:

$$V(s,\vec{q}',c) = \sum_{\vec{a}} P(\vec{a}|c,\vec{q}) \left[ R(s,\vec{a}) + \gamma \sum_{s',\vec{o},\vec{q}',c'} P(s',\vec{o}|s,\vec{a}) P(\vec{q}'|c,\vec{q},\vec{a},\vec{o}) P(c'|c) V(s',\vec{q}',c') \right]$$

Figure 11.16 illustrates a slice (similar to a dynamic Bayesian network) of a two-agent DEC-POMDP with and without a correlation device. The underlying state $s$ and observations $o_i$ are determined by the DEC-POMDP model, while the controller state $q_i$ and action $a_i$ are determined by the policy parameters.

### 5.3.2 Policy Iteration for Infinite-Horizon DEC-POMDPs

An infinite-horizon DEC-POMDP could produce infinitely many observation sequences. To approach near-optimal value, it may be necessary to increase the controller size. In fact, ε-convergence can only be guaranteed when we increase the number of controller states using an *exhaustive backup* operation. An exhaustive backup introduces new controller states for each possible action and each possible branch given an observation to existing controller states [5]. This is similar to the exhaustive backup in the finite-horizon case, except that we grow existing controllers rather than policy trees.

To formalize this process, let $Q_i^t$ denote the set of controller nodes for agent $i$ after iteration $t$. For each possible one-step policy, a new controller node is added. Thus, for each agent $i$, $|A_i||Q_i|^{|\Omega_i|}$ nodes are added to the controller. In the finite-horizon algorithm, the exhaustive backup was followed by a pruning step, eliminating dominated policy trees. Here, an analogous procedure can be used [5].

**input** : DEC-POMDP, correlated joint controller, convergence parameter $\varepsilon$
**output**: A correlated joint controller that is $\varepsilon$-optimal for all states.

```
1  begin
2      t ← 0
3      while γ^{t+1} · |R_max|/(1 − γ) > ε do
4          t ← t + 1
5          Evaluate correlated joint controller by solving a system of linear equations
6          Perform an exhaustive backup to add nodes to the local controllers
7          Perform value-preserving transformations on the controller
8      end
9      return correlated joint controller
10 end
```

**Algorithm 11.6:** Policy iteration for infinite-horizon DEC-POMDPs.

**Definition 11.10 (Value-preserving transformation)** *Given two correlated joint controllers $C$ and $D$ with node sets $\vec{Q}$ and $\vec{R}$, respectively, we say that changing controller $C$ to $D$ is a value-preserving transformation if there exist mappings $f_i : Q_i \to \Delta R_i$ for each agent $i$ and $f_c : Q_c \to \Delta R_c$ such that:*

$$V(s,\vec{q}) \leq \sum_{\vec{r}} P(\vec{r}|q)V(s,\vec{r})$$

The goal of a value-preserving transformation is to reduce the size of a controller without decreasing its value, or to improve the value without changing the size. In general, reducing the size of the controller is necessary between exhaustive backup steps because those steps increase the size of the controller in a doubly exponential manner. Several such transformations that can be implemented efficiently using linear programming have been formulated [5].

The complete policy iteration procedure, sketched in Algorithm 11.6, interleaves exhaustive backups with value-preserving transformations. Unlike single agent MDPs, there is no Bellman residual for testing convergence in this case. Therefore, it is necessary to use the discount factor $\gamma$ and the number of iterations to define a simpler $\varepsilon$-convergence test. Let $|R_{max}|$ denote the largest absolute value of a one-step reward in the DEC-POMDP. Then the algorithm terminates after iteration $t$ if $\gamma^{t+1} \cdot |R_{max}|/(1-\gamma) \leq \varepsilon$. Intuitively, the algorithm exploits the fact that due to discounting, at some point the future rewards collected are negligible.

As with optimal algorithms for finite-horizon DEC-POMDPs, producing near-optimal controllers is intractable. In practice, the value-preserving transformations cannot reduce the size of the controllers sufficiently to continue executing the algorithm until the convergence criterion is met. However, several approximate techniques for infinite-horizon DEC-POMDPs have been developed based

For variables of each agent $i$: $x(q_i, a_i)$, $y(q_i, a_i, o_i, q_i')$ and $z(\vec{q}, s)$

Maximize $\sum_s b_0(s) z(\vec{q}^0, s)$, subject to

The Bellman constraints:

$\forall \vec{q}, s \; z(\vec{q}, s) =$

$$\sum_{\vec{a}} \left( \prod_i x(q_i, a_i) \left[ R(s, \vec{a}) + \gamma \sum_{s'} P(s'|s, \vec{a}) \sum_{\vec{o}} O(\vec{o}|s', \vec{a}) \sum_{\vec{q}'} \prod_i y(q_i, a_i, o_i, q_i') z(\vec{q}', s') \right] \right)$$

And probability constraints for each agent $i$:

$$\forall q_i \sum_{a_i} x(q_i, a_i) = 1, \quad \forall q_i, o_i, a_i \sum_{q_i'} y(q_i, a_i, o_i, q_i') = 1$$
$$\forall q_i, a_i \; x(q_i, a_i) \geq 0, \quad \forall q_i, o_i, a_i \; y(q_i, a_i, o_i, q_i') \geq 0$$

Table 11.1: The NLP defining a set of optimal fixed-size DEC-POMDP controllers. For each agent $i$, variable $x(q_i, a_i)$ represents $P(a_i|q_i)$, variable $y(q_i, a_i, o_i, q_i')$ represents $P(q_i'|q_i, a_i, o_i)$, and variable $z(\vec{q}, s)$ represents $V(\vec{q}, s)$ where $\vec{q}^0$ represents the initial controller node for each agent.

on these principles by simply restricting the size of each controller and optimizing value with a *bounded* amount of memory. We discuss one such approach below.

### 5.3.3 Optimizing Fixed-Size Controllers Using Non-Linear Programming

With a fixed controller size, the problem of optimizing the joint controller can be represented as a non-linear program (NLP) by creating a set of new variables that represent the values of each set of controller states and underlying world state. Unlike the dynamic programming backups, which iteratively improve the probabilities and could get stuck in low-quality local optima, the NLP approach allows both the values and probabilities in the controller to be optimized simultaneously. While the NLP is generally harder to solve, the approach results in a search process that is more sophisticated and can leverage state-of-the-art solvers. Existing NLP solvers do not guarantee global optimality, but experimental results show that the NLP formulation is advantageous [1]. In practice, DEC-POMDPs can have small optimal controllers or can be approximated effectively with small controllers. Furthermore, the NLP approach can optimize value for a specific given initial *belief* state, thus making better use of limited controller size.

The non-linear program for a DEC-POMDP with an arbitrary number of agents is shown in Table 11.1. It optimizes the value of a set of fixed-size con-

trollers given an initial state distribution and the DEC-POMDP model. The variables for this problem are the action selection and node transition probabilities for each node of each agent's controller as well as the joint value of a set of controller nodes. Hence, these variables are for each agent $i$, $P(a_i|q_i)$ and $P(q_i'|q_i, a_i, o_i)$, and for the set of agents and any state, $V(\vec{q}, s)$. The NLP objective is to maximize the value of the initial set of nodes at the initial state distribution. The constraints include the Bellman constraints and additional probability constraints. The Bellman constraints, which are non-linear, ensure that the values are correct given the action and node transition probabilities. The probability constraints ensure that the action and node transition values are proper probabilities. It is straightforward to add a correlation device to the NLP formulation simply by adding a new variable for the transition function of the correlation device. As expected, a correlation device can improve the value achieved by the NLP approach, particularly when each controller is small [1].

# 6 Multiagent Execution

We conclude this chapter by turning to the actual execution of multiagent planning and control decisions. To the extent that the knowledge used for decision making was correct, the actual trajectory of the world state should mimic the expectations of the agents. Even in cases where actions or observations are uncertain, as is modeled in DEC-POMDPs, the planning and control decisions should have anticipated the possibilities and formulated responses to the foreseen contingencies.

Of course, the rosy picture above can fail to materialize, when the model of the world used by agents for making planning and control decisions is incorrect or incomplete relative to the agents' true world. In such situations, agents can find themselves in unanticipated states, and need to decide how to respond in the near-term, and perhaps also how to update their models so as to make better planning and control decisions in the future. In this section, we can only scratch the surface of the challenges posed in multiagent execution, and of some approaches to overcoming them.

## 6.1 Multiagent Plan Monitoring

Detecting deviations from anticipated trajectories is one challenge that is significantly more difficult in multiagent settings than single-agent settings. A single agent can use its observations to form beliefs about its current state, and then determine whether it had anticipated possibly having such beliefs at this point in its execution. If not, the agent can invoke a response that could attempt to repair its existing plan by injecting new actions that in expectation will return it to an an-

ticipated trajectory. Or the agent could treat its new beliefs about its current state as the starting point for building a new plan to achieve its objectives. See [50] for techniques of this kind.

In a multiagent system, certainly the same kind of process could occur, but now recovery is harder: an agent cannot in general inject new actions or replace its old plan with a new one without coordinating with other agents, resolving any new interagent faults that its changed plan introduces. Furthermore, it could be that a better way of recovering from such a deviation would be to have one or more other agents change their plans, even though their old plans were proceeding as expected.

Even more problematic are situations where no agent in isolation perceives a deviation from expectations (each foresaw that its current state might have arisen) but the agents collectively have reached an unexpected joint state. Detecting such a deviation requires not only that agents share local information, but that one or more agents are knowledgeable about which (partial) joint states are expected and which are not. Effectively, monitoring the execution of a multiagent plan can amount to a non-trivial collaborative problem-solving effort among the agents.

## 6.2 Multiagent Plan Recovery

In the multiagent setting, the same dilemma occurs as in single-agent planning when execution diverges from what was planned: should the agent(s) try to reuse some of the old plan(s) so as to take advantage of the effort that went into constructing them, or should the agent(s) build entirely new plan(s) to achieve their objectives from the current state. If they choose the latter route, then replanning in the multiagent context can be accomplished with any of the techniques already discussed.

Plan repair is potentially more cost-efficient, but more complicated. If only some agents need to repair their plans (the others have not deviated), repair nonetheless could involve all of the agents as the repaired plans need to be coordinated with others' unchanged plans. This could lead to others needing to change their plans, in a chain reaction of planning and coordination efforts. And this does not even account for opportunities for agents to reallocate responsibilities among themselves, such as if an agent that has deviated from expectations has exhausted a resource in its attempt to establish a condition in the world, and it *must* now fall to another agent that has reserve resources.

The preceding discussion has further made an important assumption – that the agents should treat the deviation as an anomaly rather than as an indication that the planning knowledge is itself flawed. More broadly, agents can view divergence from expectations as a learning opportunity to correct/improve the models used for planning.

A simple example of such a response can occur in the case of social laws. The designer(s) of the laws used the models of goals/rewards to identify states to avoid, and of actions to identify and prohibit bad precursor state-and-action combinations. If a state to avoid is reached nonetheless, the agents can update their transition models to build better plans/laws in the future, or might perform Q-learning to directly learn what actions not to take in particular states.

A danger in doing such learning, as discussed in Chapter 10, is that if multiple agents learn simultaneously, their local adaptations might not combine into a coherent joint adaptation. For example, if mobile robotic agents collide in a particular location, there is a danger that they *all* might now avoid that location, which would make deliveries to that location impossible. Techniques for controlling learning and adaptation in multiagent systems (see Chapter 10) is thus pertinent in this context.

## 6.3 Multiagent Continuous Planning

The extreme case of multiagent plan monitoring and repair/replanning is where agents are continually reconsidering and revising their plans. When repair/replanning is punctuated, it is not unreasonable to assume that agents can all suspend their plan executions until that process is complete, and then they proceed with executing their (new) coordinated plans unless and until another deviation occurs. This is not without problems in dynamic domains, where the environment keeps changing while agents are thinking. Thus, when planning is continuous, agents cannot afford to wait for convergence to an assuredly coordinated joint plan. Instead, agents should be able to revise and pursue plans despite those plans perhaps being (temporarily) uncoordinated.

One of the earliest examples of this form of continuous planning in multiagent systems was an approach called *partial global planning* (PGP) [20]. PGP was applied to the interpretation problem of tracking vehicles in a distributed sensor network. This kind of application is particularly forgiving of lack of coordination, which at worst only wastes computational resources. That is, the interpretation problem can be solved with functionally-accurate cooperation, as described in Section 3.2.2. This is in contrast to problems, like air traffic control, where even brief periods of mis-coordination can have catastrophic consequences.

Each sensor agent builds a plan as to how it will process its data: which data it will process in what order, when it will combine partial interpretations into larger interpretations, etc. Because agents' sensor regions can overlap, if agents can communicate about their plans, then by comparing its plans with its neighbors an agent can decide which (if any) signals in an overlapping region it should process, and which it should leave to others. Furthermore, because vehicles tend to not

disappear at boundaries, agents can help each other by providing partial interpretations near their boundaries to others, to help others focus interpretation problem solving on compatible extensions.

Of course, over time new sensor signals can arrive, or data might prove more noisy and take longer to process than expected. As a result, an agent might change its plan. If others know of this change, then they might in turn change their plans, possibly leading to chain reactions (and even cycles) of plan changes. At times these efforts can lead to significant improvements in joint behavior, but at other times the improvements might be smaller than the communication and computation overhead of attaining them. PGP combats the costs and delays in coordinating responses to such dynamics with various mechanisms, including:

1. Abstraction: As already discussed in this chapter, abstraction benefits multi-agent planning by allowing coordination decisions to be based on fundamental agent interactions without getting distracted by details of local plans. In PGP, even though each agent builds a detailed plan for processing signals and constructing larger interpretations, the agents communicate with each other only about what partial interpretations they plan to construct and when. Thus, even though internally an agent might frequently make changes to its detailed plan, other agents' perceptions of its plans generally evolve much more slowly.

2. Decentralization: PGP assumes that responsibility for coordination decisions can be distributed among agents in any of a variety of ways, as expressed in a *meta-level organization* (MLO). The MLO defines roles, protocols, and authority structures that the agents use when solving the *coordination* problem, analogously to how the agents' domain-level organization guides the agents' activities in solving the interpretation problem. In particular, the MLO can distribute coordination responsibility such that each agent has authority to modify its own plans based on its current partial view of the plans of (some of) the other agents.

3. Partial Global Reasoning: An agent forms a partial global plan by combining the abstract plans it has received from others with its own. The agent can then search through modifications to the partial global plan to find a better joint plan. For example, it might determine that various agents should reorder their tasks to change who is responsible for portions of the overlapping regions, and who will formulate which interpretations. In the PGP implementation, the search is done heuristically and in a greedy hill-climbing fashion, since finding optimal solutions would take much more time and such solutions will likely be made obsolete quickly due to further dynamics. An agent then changes its local plan according to how *it* thinks

the joint plan should change. A key assumption behind this strategy is that all agents, given the same information, would formulate the same revised partial global plan. Hence, unilateral changes to a plan in expectation that other agents will make complementary changes to their plans is warranted given sufficient propagation of planning information based on the MLO.

4. Communication Planning: By examining the partial global plan, an agent can determine when an interpretation will be formed by one agent that could be of interest to another agent, and can explicitly plan the communication action to transmit the result. If results need to be integrated into a larger partial interpretation, an agent using PGP will construct a tree of exchanges such that, at the root of the tree, partially integrated results will be at the same agent, who can then construct the complete result.

5. Asynchrony: Agents asynchronously adapt their local plans and communication plans to their partial global views. This sacrifices global consistency across plans for rapid responsiveness to changing awareness. Depending on the MLO and the relative domain dynamics, the agents could ultimately converge on consistent joint plans, but PGP allows each agent to pursue its best guess as to what its local plan should be at any given time. If it guesses wrong, its efforts were wasted, but had it idled waiting for consistent plans those cycles would have gone to waste anyway.

6. Dampened Responsiveness: As an agent's plan changes in the dynamic environment, it can determine how its abstract plan that it has told other agents about has been affected, if at all. Some changes to a local plan might, for example, delay the formation and transmission of a partial interpretation. The agent must determine whether a change to its abstract plan is worth telling other agents about. In PGP, a simple thresholding strategy was used: if an abstract plan step will occur earlier or later than expected by more that a parameterized number of time steps, then the agent should alert others. A larger threshold effectively introduced "slack" into the system, cutting down on coordination overhead by allowing greater degrees of interagent plan slippage. The parameter would be tuned empirically.

Advances in continual planning in multiagent systems have extended and refined these types of strategies over the years. For example, the DARPA Coordinators program [35] emphasized helping teams of humans who are distributed geographically to manage and time their activities so as to achieve joint objectives. One research thrust for this problem was for the teams to each represent a space of possible schedules to follow, where unfolding events can narrow the space of remaining possibilities, and where a threshold defined on this process determines

whether an agent should inform others about such changes [3]. Another was for agents to explicitly model the probabilities of satisfying their contributions to joint objectives, and updating each other as these probabilities decreased significantly [43].

# 7 Conclusions

Multiagent planning and control with more agents, capable of more behaviors, operating in uncertain and partially-observable worlds, introduces and compounds daunting computational challenges. Research has sought to exploit structure in problems that allow solutions to be composed from solutions to localized subproblems, and this chapter has illustrated various strategies for different types of problem structures and performance requirements. Significant progress has been made, and yet substantial challenges remain. We conclude by summarizing other important past and ongoing work in this area.

Multiagent planning has been studied since the founding of the field of distributed AI. Some of the earliest work in this area includes that of Georgeff [26, 27], who developed some of the earliest multiagent plan deconfliction techniques, and of Corkill [14], who developed a distributed version of the NOAH planner created by Sacerdoti [51]. Corkill and colleagues, especially Lesser, pioneered the use of organizational techniques for multiagent control [16]. Decker and Lesser generalized techniques for coordinating agent plans in their work on GPGP, and for representing complex multiagent task networks in their work on TAEMS [39].

Planning for teams of agents was investigated not only by Tambe [49, 59] (Section 4.3), but also by Grosz and Kraus [31], building on concepts from Cohen and Levesque [12]. Multiagent planning and scheduling, involving dealing with temporal constraints, also has a rich literature (e.g., [8, 61]). Other work for coordinating plans that agents largely form separately includes that of Tonino et al. [60].

Other techniques that have formulated the multiagent planning problem in decision-theoretic terms include those that solve problems where agents interact through the assignment (and reassignment) of resources [19, 66], and where agents interact by changing shared state in structured ways that enable each other [4, 64].

Memory-bounded dynamic programming (MBDP) [53] has been dramatically improved in recent years by introducing a variety of methods to reduce the number of observations considered by the policy, and employing efficient pruning techniques. Point-based methods have been recently introduced to cope with the NP-hardness of the backup operation [37]. This algorithm exploits recent advances

in the weighted CSP literature to offer a polytime approximation scheme that can handle a much larger number of belief points (*MaxTrees*). Another technique, trial-based dynamic programming (TBDP) [65], combines the main advantages of DP-JESP with MBDP to avoid the expensive backup operations, allowing problems with much larger state spaces to be tackled.

The locality of agent interaction – the fact that each agent interacts with a small number of neighboring agents – has proved crucial to the development of DEC-POMDP algorithms that can handle dozens of agents. Specialized models such as network distributed POMDPs (ND-POMDPs) have been introduced to capture structured interactions and develop early algorithms that can exploit such structures [45]. More recently, the constrained-based dynamic programming (CBDP) algorithm has been shown to provide magnitudes of speedup thanks to its linear complexity in the number of agents [36]. Algorithms for solving loosely-coupled infinite-horizon problems have also been developed. One promising direction is based on transforming the policy optimization problem to that of likelihood maximization in a mixture of dynamic Bayesian networks [38]. Based on this reformulation, the expectation-maximization (EM) algorithm has been used to compute the policy via a simple message-passing paradigm guided by the agent interaction graph.

# 8 Exercises

1. Level 1 Analyze a multiperson problem that you have been involved in solving. Identify localities in the structure of the problem, and strategies for composing an overall solution from solutions to the localized subproblems.

2. Level 1 Do you agree with the stance taken in this chapter that multiagent planning requires both that the plan formulation process be distributed among agents and that the resulting plan construct be distributed as well? If so, justify in your own words why you believe this stance is warranted. If not, give counterexamples to this stance and justify why they arguably embody multiagent planning.

3. Level 2 The broad brushstrokes of social laws to avoid collisions between robots operating in a gridworld was provided in this chapter. This question asks you to elaborate a bit.

   (a) For an 8 x 8 gridworld (empty other than robots), flesh out an example of the social laws that avoid collisions, indicating for each location where a robot is allowed to go, along with any other laws that the robots should follow.

(b) Are there gridworld shapes (again, empty other than robots) for which useful social laws cannot be constructed? If so, give examples, and if not, explain why not.

(c) Now, say in the 8 x 8 gridworld there is a wall that partitions the environment in half except there is one pair of cells on each side that are connected. How would you build social laws to handle such a case, assuming that a robot might need to visit locations on both sides of the world?

4. Level 4 Implement a simulation of a robot gridworld. Create in that world simulated robots, and endow them with the capabilities and constraints associated with Traffic Law 2 in [56]. Experimentally investigate the performance of the robots following that traffic law to determine how well they perform as the number of robots increases in a fixed-size grid.

5. Level 2 Describe and analyze a real-world (human) instance of organizational redesign. What alternative decompositions of the larger problem into interacting roles were possible, and why was the particular choice made during the redesign? Develop a specification for the space of designs, and suggest a search strategy for finding and implementing a good design from that space. Is your search assured to return an optimal solution? Is it efficient?

6. Level 1 Consider the contract net protocol where announcements can be either about tasks that need to be done or the availability of resources that could be assigned tasks.

(a) Name a real-life example where task announcement makes much more sense than availability announcement. Justify why.

(b) Now name a real-life example where availability announcement makes much more sense. Justify why.

(c) Let's say that you are going to build a mechanism that oversees a distributed problem-solving system, and can "switch" it to either a task or availability announcement mode.

i. Assuming communication costs are negligible, what criteria would you use to switch between modes? Be specific about what you would test.

ii. If communication costs are high, now what criteria would you use? Be specific about what you would test.

7. Level 2/3 We noted that task announcing can be tricky: If a manager is too fussy about eligibility, it might get no bids, but if it is too open it might have to process too many bids, including those from inferior contractors. Let us say that the manager has $n$ levels of eligibility specifications from which it needs to choose one. Describe how it would make this choice based on a decision-theoretic formulation. How would this formulation change if it needed to consider competition for contractors from other managers?

8. Level 2 A folk theorem in the organization literature is that in human organizations, task decompositions invariably lead to clear assignments of subtasks to members of the organization. Give an example of where decomposition without look-ahead to available contractors can be detrimental. Give an example where biasing decomposition based on available contractors can instead be detrimental. Finally, give an algorithm for alternating between decomposition and assignment to incrementally formulate a distributed problem-solving system. Is your algorithm assured of yielding an optimal result? Is it complete?

9. Level 1 Consider the pursuit task, with four predators attempting to surround and capture a prey. Define an organizational structure for the predators. What are the roles and responsibilities of each? How does the structure indicate the kinds of communication patterns (if any) that will lead to success?

10. Level 2 Consider the following simple instance of the distributed delivery task. Robot $A$ is at position $\alpha$ and robot $B$ is at position $\beta$. Article $X$ is at position $\xi$ and needs to go to position $\psi$, and article $Y$ is at position $\psi$ and needs to go to $\zeta$. Positions $\alpha$, $\beta$, $\xi$, $\psi$, and $\zeta$ are all different.

    (a) Define in STRIPS notation, suitable for partial-order planning, simple operators Pickup, Dropoff, PickDrop, and Return, where Pickup moves the robot from its current position to a pickup position where it then has the article associated with that position; Dropoff moves a robot and an article it holds to a dropoff position where it no longer has the article; PickDrop combines the two (it drops off its article and picks up another associated with that position); and Return moves a robot back to its original position.

    (b) Using these operators, generate the partial-order plan with the fewest plan steps to accomplish the deliveries. Decompose and distribute this plan to the robots for parallel execution, inserting any needed synchronization actions. How does the use of multiple robots affect the plan execution?

(c) Using the operators, generate the partial-order plan that, when distributed, will accomplish the deliveries as quickly as possible. Is this the same plan as in the previous part of this problem? Why or why not?

11. Level 2 Given the previous problem, include in the operator descriptions conditions that disallow robots to be at the same position at the same time (for example, a robot cannot do a pickup in a location where another is doing a dropoff). Assuming each robot was given the task of delivering a different one of the articles, generate the individual plans and then use the plan modification algorithm to formulate the synchronized plans, including any synchronization actions into the plans. Show your work.

12. Level 2 Consider the delivery problem given before the previous problem. Assume that delivery plans can be decomposed into 3 subplans (pickup, dropoff, and return), and that each of these subplans can further be decomposed into individual plan steps. Furthermore, assume that robots should not occupy the same location at the same time – not just at dropoff/pickup points, but throughout their travels. Use the hierarchical behavior-space search algorithm to resolve potential conflicts between the robots' plans, given a few different layouts of the coordinates for the various positions (that is, where path-crossing is maximized and minimized). What kinds of coordinated plans arise depending on at what level of the hierarchy the plans' conflicts are resolved through synchronization?

13. Level 3 Assume that distributed delivery robots are in an environment where delivery tasks pop up dynamically. When a delivery needs to be done, the article to be delivered announces that it needs to be delivered, and delivery agents within a particular distance from the article hear the announcement.

    (a) Assume that the distance from which articles can be heard is small. What characteristics would an organizational structure among the delivery agents have to have to minimize the deliveries that might be overlooked?

    (b) Assume that the distance is instead large. Would an organizational structure be beneficial anyway? Justify your answer.

    (c) As they become aware of deliveries to be done, delivery agents try to incorporate those into their current delivery plans. But the dynamic nature of the domain means that these plans are undergoing evolution. Under what assumptions would partial global planning be a good approach for coordinating the agents in this case?

(d) Assume you are using partial global planning for coordination in this problem. What would you believe would be a good planning level for the agents to communicate and coordinate their plans? How would the agents determine whether they were working on related plans? How would they use this view to change their local plans? Would a hill-climbing strategy work well for this?

14. Level 1 Given a flawed multiagent plan $M$ with more than one unflagged causal link to adjust, which causal link should the plan modification algorithm prefer to adjust? Justify your (heuristic) selection strategy.

15. Level 1 Give an example of a real-world situation in which multiple agents operate under partial observability and each agent has access to *different* partial information about the overall state. Can agents share all their knowledge all the time in your example? If yes, explain how. If not, explain why.

16. Level 2 The DEC-POMDP model (Definition 11.7) does not include explicit communication. Suppose that each agent can broadcast certain messages to all the other agents in each action cycle. Define precisely this kind of a DEC-POMDP with *two* agents and explain why it is *not* an extension of the standard model (i.e., show that every DEC-POMDP with explicit communication can be reduced to a standard DEC-POMDP).

17. Level 3 The communication model presented in the previous question allows each agent to broadcast a message to all the other agents in each step. This means that the space of possible *joint* messages received by each agent grows exponentially with the number of agents. Consider a more scalable communication model that allows only one agent to broadcast a message in each cycle (e.g, when multiple agents try to broadcast messages simultaneously, this may either result in failure or success of just one agent). Define precisely one such model and determine whether it is reducible to a standard DEC-POMDP or not.

18. Level 2 Consider the complete specification of the multiagent tiger problem shown in Table 11.2.

    (a) Derive the values of the deterministic policies for horizons 1–3 shown in Figure 11.9.

    (b) Derive the values of the deterministic finite-state controller policies shown in Figure 11.11.

Tiger observation table

| Joint action | State | hl | hr |
|---|---|---|---|
| ⟨L,L⟩ | TL | 0.85 | 0.15 |
| ⟨L,L⟩ | TR | 0.15 | 0.85 |
| ⟨O*,*⟩ | * | 0.5 | 0.5 |
| ⟨*,O*⟩ | * | 0.5 | 0.5 |

Tiger transition table

| Joint action | Current state | Next state | Probability |
|---|---|---|---|
| ⟨L,L⟩ | TL | TL | 1.0 |
| ⟨L,L⟩ | TR | TR | 1.0 |
| ⟨O*,*⟩ | TL | TL | 0.5 |
| ⟨O*,*⟩ | TL | TR | 0.5 |
| ⟨*,O*⟩ | TR | TR | 0.5 |
| ⟨*,O*⟩ | TR | TL | 0.5 |

Tiger reward table

| Joint action | State | Value | Joint action | State | Value | Joint action | State | Value |
|---|---|---|---|---|---|---|---|---|
| ⟨L,L⟩ | * | -2 | ⟨OR,L⟩ | TR | -101 | ⟨OL,L⟩ | TR | 9 |
| ⟨L,OR⟩ | TR | -101 | ⟨OR,L⟩ | TL | 9 | ⟨OL,L⟩ | TL | -101 |
| ⟨L,OR⟩ | TL | 9 | ⟨OR,OR⟩ | TR | -50 | ⟨OL,OR⟩ | * | -100 |
| ⟨L,OL⟩ | TR | 9 | ⟨OR,OR⟩ | TL | 20 | ⟨OL,OL⟩ | TR | 20 |
| ⟨L,OL⟩ | TL | -101 | ⟨OR,OL⟩ | * | -100 | ⟨OL,OL⟩ | TL | -50 |

Table 11.2: Tiger observation, transition, and reward tables.

19. Level 2 In the multiagent tiger problem, suppose that the reward for opening the correct door (e.g., <OR,OR> when the state is TR) is increased to 50. Is the horizon 1 policy in Figure 11.9 still optimal? If not, what is the optimal policy (and its value)? Repeat the question for horizons 2 and 3.

20. Level 2 In the multiagent tiger problem, the optimal policy is to listen for several steps before opening any door. If the observation probabilities increase from 0.85 to 0.9, does that change the optimal horizon 1 policy? What about horizons 2 and 3?

21. Level 2/3 If all agents share their observations with each other at each step, the problem becomes centralized. In the multiagent tiger problem, what would the resulting observations (and their probabilities) be for each agent when observations are shared? How does this change the optimal policies for horizons 1 and 2? Would the agents ever choose to open different doors? Is a centralized solution (with shared observations) always guaranteed to have value at least as high as a decentralized solution?

22. Level 2/3 If the transition and observation probabilities are independent for each agent and the reward values are additive between the agents, the problem can be solved as a set of independent problems whose solutions can be summed together. In the multiagent tiger problem, the observations are

independent, but the transitions and rewards depend on all agents. Consider the case where the tiger does not transition after a door is opened and each agent receives a reward of 10 for opening the correct door, $-50$ for opening the incorrect door and $-1$ for listening. What are the optimal horizon 1, 2, and 3 policies for this case?

23. Level 2/3 Given the same number of nodes, stochastic controllers often allow higher-valued policies to be constructed compared to deterministic controllers. Is there a one-node stochastic controller with a higher value than the optimal one-node deterministic controller in the multiagent tiger problem? If there is, construct one. Otherwise, prove that this is not possible in this case.

# Acknowledgments

We thank Christopher Amato for invaluable feedback on the presentation of decentralized POMDPs, and for his help generating the corresponding examples, figures, and exercises. Jeffrey Cox's work on multiagent partial-order causal-link planning has been drawn on heavily in this chapter as well. We thank all our former students and collaborators for the many fruitful discussions on the topics covered in this chapter, which helped sharpen our understanding of multiagent planning.

# References

[1] Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Autonomous Agents and Multi-Agent Systems*, 21(3):293–320, 2010.

[2] Christer Bäckström. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, 9:99–137, 1998.

[3] Laura Barbulescu, Zack Rubinstein, Stephen Smith, and Terry Lyle Zimmerman. Distributed coordination of mobile agent teams: The advantage of planning ahead. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 1331–1338, 2010.

[4] Raphen Becker, Victor Lesser, and Shlomo Zilberstein. Decentralized Markov decision processes with event-driven interactions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, volume 1, pages 302–309, 2004.

[5] Daniel S. Bernstein, Christopher Amato, Eric A. Hansen, and Shlomo Zilberstein. Policy iteration for decentralized control of Markov decision processes. *Journal of Artificial Intelligence Research*, 34:89–132, 2009.

[6] Daniel S. Bernstein, Eric A. Hansen, and Shlomo Zilberstein. Bounded policy iteration for decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1287–1292, 2005.

[7] Daniel S. Bernstein, Shlomo Zilberstein, and Neil Immerman. The complexity of decentralized control of Markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 32–37, 2000.

[8] Aurélie Beynier and Abdel-Illah Mouaddib. A polynomial algorithm for decentralized Markov decision processes with temporal constraints. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-05)*, pages 963–969, 2005.

[9] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the Conference on Theoretical Aspects of Rationality and Knowledge (TARK-96)*, pages 195–210, 1996.

[10] Kathleen M. Carley and Les Gasser. *Computational Organization Theory*. MIT Press, 1999.

[11] Bradley J. Clement, Edmund H. Durfee, and Anthony C. Barrett. Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research*, 28:453–515, 2007.

[12] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2-3):213–261, 1990.

[13] Susan E. Conry, Kazuhiro Kuwabara, Victor R. Lesser, and Robert A. Meyer. Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-21(6):1462–1477, 1991.

[14] Daniel Corkill. Hierarchical planning in a distributed problem-solving environment. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 168–175, 1979.

[15] Daniel Corkill. Blackboard systems. *AI Expert*, 6(9), January 1991.

[16] Daniel Corkill and Victor Lesser. The use of meta-level control for coordination in a distributed problem solving network. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 748–756, 1983.

[17] Daniel D. Corkill. *A Framework for Organizational Self-Design in Distributed Problem Solving Networks*. PhD thesis, University of Massachusetts, 1982.

[18] Randall Davis and Reid Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.

[19] Dmitri A. Dolgov and Edmund H. Durfee. Resource allocation among agents with MDP-induced preferences. *Journal of Artificial Intelligence Research*, 27:505–549, 2006.

[20] Edmund H. Durfee. *Coordination of Distributed Problem Solvers*. Kluwer Academic Press, 1988.

[21] Edmund H. Durfee. Organizations, plans, and schedules: An interdisciplinary perspective on coordinating AI systems. *Journal of Intelligent Systems*, 3(2-4):157–187, 1993.

[22] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Cooperation through communication in a distributed problem solving network. In M. Huhns, editor, *Distributed Artificial Intelligence*, chapter 2. Pitman, 1987.

[23] Eithan Ephrati and Jeffrey S. Rosenschein. Divide and conquer in multi-agent planning. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 375–380, 1994.

[24] Eithan Eprhati, Martha Pollack, and Jeffrey S. Rosenschein. A tractable heuristic that maximizes global utility through local plan combination. In *Proceedings of the First International Conference on Multiagent Systems (ICMAS-95)*, pages 94–101, 1995.

[25] Michael Fisher and Michael Wooldridge. Distributed problem-solving as concurrent theorem-proving. In *Proceedings of MAAMAW'97*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1997.

[26] Michael Georgeff. Communication and interaction in multi-agent planning. In *Proceedings of the 3rd National Conference on Artificial Intelligence (AAAI-83)*, pages 125–129, 1983.

[27] Michael P. Georgeff. A theory of action for multiagent planning. In *AAAI*, pages 121–125, 1984.

[28] Piotr J. Gmytrasiewicz and Prashant Doshi. A framework for sequential planning in multiagent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.

[29] Claudia Goldman and Jeffrey S. Rosenschein. Emergent coordination through the use of cooperative state-changing rules. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 408–413, 1994.

[30] Claudia V. Goldman and Shlomo Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22:143–174, 2004.

[31] Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.

[32] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 709–715, 2004.

[33] Nick R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8:223–250, 1993.

[34] Subbarao Kambhampati, Mark Cutkosky, Marty Tenenbaum, and Soo Hong Lee. Combining specialized reasoners and general purpose planners: A case study. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI-91)*, pages 199–205, 1991.

[35] Robert Kohout. The DARPA COORDINATORS program: A retrospective. In *Proceedings of the 2011 International Conference on Collaborative Technologies and Systems (CTS-10)*, page 342, 2011.

[36] Akshat Kumar and Shlomo Zilberstein. Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, pages 561–568, 2009.

[37] Akshat Kumar and Shlomo Zilberstein. Point-based backup for decentralized POMDPs: Complexity and new algorithms. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-10)*, pages 1315–1322, 2010.

[38] Akshat Kumar and Shlomo Zilberstein. Message-passing algorithms for large structured decentralized POMDPs. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-11)*, pages 1087–1088, 2011.

[39] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. Nagendra Prasad, A. Raja, R. Vincent, P. Xuan, and X.Q̃. Zhang. Evolution of the GPGP/TAEMS domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, 2004.

[40] Victor R. Lesser and Daniel D. Corkill. Functionally accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11:81–96, 1981.

[41] Victor R. Lesser and Lee D. Erman. Distributed interpretation: A model and an experiment. *IEEE Transactions on Computers*, C-29(12):1144–1163, 1980.

[42] Douglas MacIntosh, Susan Conry, and Robert Meyer. Distributed automated reasoning: Issues in coordination, cooperation, and performance. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-21(6):1307–1316, 1991.

[43] Rajiv T. Maheswaran, Craig Milo Rogers, Romeo Sanchez, and Pedro A. Szekely. Decision-support for real-time multi-agent coordination. In *AAMAS'10*, pages 1771–1772, 2010.

[44] Ranjit Nair, David Pynadath, Makoto Yokoo, Milind Tambe, and Stacy Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multi-agent settings. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 705–711, 2003.

[45] Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 133–139, 2005.

[46] Raz Nissim, Ronen I. Brafman, and Carmel Domshlak. A general, fully distributed multi-agent planning algorithm. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 1323–1330, 2010.

[47] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.

[48] David V. Pynadath and Milind Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423, 2002.

[49] David V. Pynadath and Milind Tambe. An automated teamwork infrastructure for heterogeneous software agents and humans. *Autonomous Agents and Multi-Agent Systems*, pages 71–100, 2003.

[50] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2nd edition, 2003.

[51] Earl Sacerdoti. *A Structure for Plans and Behavior*. Elsevier North-Holland Inc., 1977.

[52] Sandip Sen and Edmund H. Durfee. A contracting model for flexible distributed scheduling. *Annals of Operations Research*, 65:195–222, 1996.

[53] Sven Seuken and Shlomo Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2009–2015, 2007.

[54] Sven Seuken and Shlomo Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.

[55] Yoav Shoham and Moshe Tennenholtz. On the synthesis of useful social laws for artificial agent societies. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 276–281, 1992.

[56] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73:231–252, 1995.

[57] Mark Sims, Daniel Corkill, and Victor Lesser. Automated organization design for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 16(2):151–185, 2008.

[58] Daniel Szer and Francois Charpillet. Point-based dynamic programming for DEC-POMDPs. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, pages 1233–1238, 2006.

[59] Milind Tambe. Agent architectures for flexible, practical teamwork. In *Proceedings of AAAI/IAAI'97*, pages 22–28, 1997.

[60] Hans Tonino, André Bos, Mathijs de Weerdt, and Cees Witteveen. Plan coordination by revision in collective agent based systems. *Artificial Intelligence*, 142(2):121–145, 2002.

[61] Ioannis Tsamardinos, Martha E. Pollack, and John F. Horty. Merging plans with quantitative temporal constraints, temporally extended actions, and conditional branches. In *Proceedings of the Conference on AI Planning Systems*, pages 264–272, 2000.

[62] Guandong Wang, Weixiong Zhang, Roger Mailler, and Victor Lesser. Analysis of negotiation protocols by distributed search. In Victor Lesser, Charles Ortiz, and Milind Tambe, editors, *Distributed Sensor Networks: A Multiagent Perspective*, pages 339–361. Kluwer Academic Publishers, 2003.

[63] Daniel S. Weld. An introduction to least commitment planning. *Artificial Intelligence Magazine*, 15(4):27–61, 1994.

[64] Stefan J. Witwicki and Edmund H. Durfee. Influence-based policy abstraction for weakly-coupled Dec-POMDPs. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)*, pages 185–192, 2010.

[65] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. Trial-based dynamic programming for multi-agent planning. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI-10)*, pages 908–914, 2010.

[66] Jianhui Wu and Edmund H. Durfee. Resource-driven mission-phasing techniques for constrained agents in stochastic environments. *Journal of Artificial Intelligence Research*, 38:415–473, 2010.

[67] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10:673–685, 1998.

[68] Chengqi Zhang. Cooperation under uncertainty in distributed expert systems. *Artificial Intelligence*, 56(1):21–69, 1992.