Approximate planning under uncertainty in partially observable environments

Matthijs T. J. Spaan

Approximate planning under uncertainty in partially observable environments

Academisch Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. mr. P. F. van der Heijden
ten overstaan van een door het college voor promoties ingestelde
commissie, in het openbaar te verdedigen in de Aula der Universiteit
op vrijdag 20 oktober 2006, te 10:00 uur

door

Matthijs Theodor Jan Spaan

geboren te Geleen

Promotiecommissie

Promotor:	Prof. dr. ir. F. C. A. Groen
Co-promotor:	Dr. N. Vlassis
Overige leden:	Prof. drs. M. Boasson
	Prof. dr. H. J. Kappen
	Prof. dr. M. van Lambalgen
	Dr. M. L. Littman
	Dr. M. A. Wiering

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

This work has been supported by PROGRESS, the embedded systems research program of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW, project AES.5414.

Advanced School for Computing and Imaging

This work was carried out in the ASCI graduate school.
ASCI dissertation series number 131.

Cover illustration: Patrick Borgers
ISBN-10: 90-74840-79-5
ISBN-13: 978-90-74840-79-8

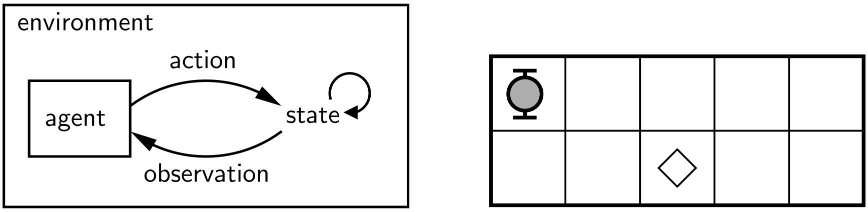
Contents

Chapter 1

Introduction

A major goal of artificial intelligence (AI) is to build *agents*: systems that perceive their environment and execute actions. For instance, a mobile robot might have a camera to observe its surroundings and wheels to move around. In particular, AI aims to develop *intelligent* agents, which attempt to perform an assigned task as well as possible. An agent is a general concept, which includes robots, intelligent computer programs, and humans. We can characterize an agent by its sense–think–act loop: it uses sensors to observe the environment, considers this information to decide what to do, and executes the chosen action. The agent influences its environment by acting and can detect the effect of its actions by sensing: the environment closes the loop. Fig. 1.1(a) shows a diagram of an agent and its operating loop in an environment.

An intelligent agent should be able to perform a given task autonomously. A human user defines the task, but the agent should act without further external control by the human. In the early days of AI, the focus was on tackling abstract problems that (appear to) require high-level cognitive skills to solve, for instance playing chess or automatic proving of theorems. However, when trying to implement such skills in a real-world system, it turns out that low-level human skills, such as recognizing objects using a camera, are major hurdles for a computer system. As such, when building an intelligent agent, one is confronted with many scientific and engineering challenges. In this thesis, we will focus on one such challenge: how should agents *act under uncertainty.* A real-world system such as a robot has to be able to deal with uncertainty from numerous sources. A major source of uncertainty for a robot are its sensors, which are often noisy and have only a limited view of the environment. A robot is also often uncertain about the effect that executing an action has on its environment. For instance, when driving through a corridor in an office building, the robot has to deal with issues like potential wheel slip or unforeseen obstacles. We will



(a) Diagram of an agent.

(b) A planning problem.

Figure 1.1: (a) A schematic representation of an agent in an environment. The agent influences the state of the environment by acting, and perceives the environment through observations. (b) A simple planning problem, in which agent located in the top-left corner has to reach a certain goal location, indicated by ⋄.

present algorithms that allow an agent to handle these kinds of uncertainty in a principled way. We will consider single-agent problems as well as multiagent settings, in which multiple agents cooperate to achieve a common task.

1.1 Planning

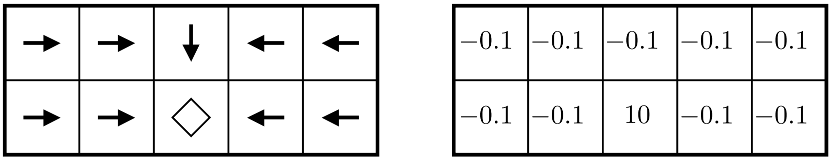
In this thesis we will consider the problem of decision making for intelligent agents: how should an agent act to perform its assigned task as well as possible. In particular, we are interested in computing *plans* for agents. Put simply, a plan tells an agent what to do in order to reach a certain goal, i.e., what action to take at each time step. Planning considers the problem of *sequential* decision making, since reaching a certain goal might require the agent to execute a particular sequence of actions. For instance, consider the simple planning problem depicted in Fig. 1.1(b). A robot in a room has to reach a goal location as fast as possible. The goal is indicated by ⋄, and the robot starts in the top-left corner. At each time step, the agent can observe its location and choose to move one square north, east, south, or west. The robot cannot reach the goal in one step, and has to plan ahead, considering multiple time steps. Given this problem description, three optimal plans exist: east-east-south, east-south-east and south-east-east. These types of planning problems are tackled in classic AI planning: find a sequence of actions that takes the environment from a starting state to a goal state (Fikes and Nilsson, 1971; Korf, 1987).

However, plans consisting of a fixed sequence of actions have only a limited applicability. Consider, for instance, the case in which our robot can start in any of the ten locations, instead of always starting from a fixed position. If we assume that the robot remains stationary when it tries to move outside its environment, a fixed-sequence plan exists that will allow the robot to reach its goal location from any starting position. However, such a plan is far from optimal and will often take more actions than necessary. For instance, move east four times, move south, followed by two times west ensures that the robot always reaches its goal. A more practical solution would be to compute an optimal sequence for each possible starting position, and let the robot's plan be *conditional* on the robot's initial state: select the shortest path for the particular starting location. Such a conditional plan ensures that the robot will reach the goal location as fast as possible from any starting location. However, until now we have assumed that the robot's motion is perfect: when the robot moves east it is always transported one square east (unless it hits the wall). In reality, robot motion is not perfect, and there will be uncertainty regarding the effect of the robot's actions.

1.2 Uncertainty

Uncertainty is abundant in real-world planning problems. A major source of uncertainty is the fact that planning is always performed in a model of the environment. A model will always be an abstraction of the real world, and the (expected) discrepancies between the model and the real world lead to uncertainty. For instance, when modeling a robot's motion, numerous parameters such as the amount of wheel slip are hard to estimate. The question is how to compute plans for agents given an uncertain model. One solution is to dispense with model-based planning altogether, and only react directly on sensory input, as "the world is its own best model" (Brooks, 1990). However, that will not be sufficient for the high-level tasks we will attempt to solve, and therefore we choose to explicitly capture uncertainty in our model. In our planning context, the dominant paradigm for representing uncertainty is probability theory (Papoulis, 1991; Bertsekas and Tsitsiklis, 2002). Other techniques for tackling uncertainty have been proposed in the AI literature, for instance Dempster-Shafer theory (Dempster, 1968; Shafer, 1976) or possibility theory (Zadeh, 1978). In a way, they extend the probabilistic framework by defining an uncertain event with two numbers, an upper and lower bound on its probability, which allows for specifying ignorance or vagueness regarding uncertain knowledge. While such alternative models of uncertainty are valuable in for instance expert systems, they are not well suited for planning (Russell and Norvig, 2003).

Returning to our example problem of Fig. 1.1(b), we can now capture the robot's imperfect motion by the following model: suppose that if the robot intends to move one square in a particular direction, it only succeeds 80% of the cases. Due to wheel slip, it will remain stationary in 10% of the cases, and in the remaining 10% the robot will overshoot and move two squares in the intended direction. As the outcome of executing a series of actions can no longer be predicted with full certainty, imperfect motion complicates our planning problem.



→	→	↓	←	←
→	→	◇	←	←

(a) Conditional plan.

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

(b) Reward model.

Figure 1.2: Example problem introduced in Fig. 1.1(b). (a) A plan that specifies for each location the action a robot should take in order to reach the ◇ goal location as fast as possible. (b) Reward model, which shows the immediate reward for each of the 10 states.

Plans consisting of fixed action sequences will no longer suffice. Instead a plan should indicate the optimal action at every location, i.e., the plan is conditional on the robot's location. In general, we want an agent to be able to respond correctly to every possible situation it might encounter. Fig. 1.2(a) shows such a conditional plan, in which the arrow at each location indicates the action the robot should take. Using this plan, when the robot accidentally overshoots due to motion noise, it can simply take the action specified at its resulting location.

Apart from actuator noise, the robot is also likely to suffer from uncertainty in its sensors. In the discussion so-far, we have assumed that the robot can detect its own location in the room with full certainty, which is not a realistic assumption for two main reasons. An agent's sensors might only provide a limited view of the environment, for instance our robot might be able to detect objects in its current room, but not in rooms adjacent to it. A second source of sensor uncertainty is noise, which is common in real-world sensors. If a robot uses a camera image to locate an object (with respect to itself), it might be able to get a reliable bearing estimate, but the exact distance of the object to the robot is much harder to estimate accurately. As for the actuator noise, we will also capture an agent's imperfect sensors using a probabilistic model. The combined uncertainty in sensors and actuators will result in uncertainty in the robot's localization. Uncertainty regarding the robot's location complicates the direct application of plans that condition on this location, such as the one depicted in Fig. 1.2(a). However, in the framework of decision-theoretic planning, techniques have been developed to cope with actuator and sensor uncertainty.

1.3 Decision-theoretic planning

We have shown why our robot should be able to act under uncertainty. Taking decisions when there is uncertainty regarding the exact consequence of a particular decision has been formalized in decision theory (Luce and Raiffa, 1957).

Decision theory considers the problem of one-step decision making, which in our case would correspond to choosing a complete (conditional) plan. After executing one particular plan, the agent will be in a certain state, and it will have preference relation over all outcome states. In our robot example possible outcomes are the ten locations of the robot, and the robot prefers the goal location above all other locations. For every possible plan, we need to know the probability that the plan will lead to a certain outcome. Given the preference relation and the probabilities, we can compute the expected value of a plan, and the agent selects the plan with the maximum expected value. For instance, our robot will choose a plan that has a high probability of leading it to the goal location quickly.

Decision theory provides a way to select among multiple plans with uncertain outcome states, but does not tackle the problem of how to compute such plans. Hence, techniques from classic AI planning, which compute plans in deterministic environments to move the agent from a start state to a goal state, have been combined with decision theory (Feldman and Sproull, 1977). In the operations-research community a different framework for planning under uncertainty has been developed, based on Markov decision processes (MDPs) (Bellman, 1957; Puterman, 1994). Instead of focusing on finding a plan which takes the agent from a start state to a goal state, in an MDP the agent receives a *reward* signal for every action it takes. The reward associated with a particular action depends on the state in which the agent took the action. The goal of the agent is now to maximize its long-term cumulative reward, i.e., the sum of rewards it expects to receive in the future. To give some intuition, Fig. 1.2(b) encodes the task of moving to the goal state as fast as possible, according to the following specifications: first of all, the robot receives a large positive reward of 10 when it has reached the goal location, in order to entice it to go to the desired location. As we would like the robot to minimize the number of actions before reaching the goal, we penalize all other actions by reward -0.1. As each superfluous action will decrease the total sum of rewards, the robot will prefer plans that minimize the expected number of steps to reach the goal location.

In general, MDPs provide a flexible framework for encoding decision-making problems involving uncertainty, and form a basis for a vast body of work in decision-theoretic planning (Boutilier, Dean, and Hanks, 1999) and reinforcement learning (Sutton and Barto, 1998). Blythe (1999) and Boutilier et al. (1999) provide overviews of decision-theoretic planning, and discuss methods based on classic AI planners as well as MDP-based techniques. In this thesis we adopt (extensions of) the MDP framework as our planning paradigm. The MDP model as described above allows for planning in stochastic environments, but assumes the agent knows the state of the environment with full certainty. As we have seen in Sec. 1.2, due to limited sensing capabilities an agent might only receive partial information regarding the system's state. In order to tackle sensor uncertainty, partially observable MDPs (POMDPs) have been developed.

As with fully observable MDPs, POMDPs originate from operations research (Sondik, 1971; Lovejoy, 1991), but in the last decade they have gained popularity in the AI community as well (Kaelbling, Littman, and Cassandra, 1998). Both MDPs and POMDPs tackle single-agent planning problems, but have been extended to cooperative multiagent systems, in which multiple agents inhabiting the same environment have to cooperate to achieve an assigned task. We will consider planning for teams of agents in the decentralized POMDP (DEC-POMDP) model (Bernstein, Givan, Immerman, and Zilberstein, 2002).

In this thesis we will present experimental results from typical AI applications such as robot navigation, but decision-theoretic planning in general and POMDPs in particular have a much wider applicability. For example, POMDPs have been considered for industrial applications such as machine maintenance (Smallwood and Sondik, 1973) or inventory control (Treharne and Sox, 2002), and also for robotic problems (Pineau, Montemerlo, Pollack, Roy, and Thrun, 2003b), man-machine interaction (Williams, Poupart, and Young, 2005b), marketing (Rusmevichientong and Van Roy, 2001), and medical applications (Hauskrecht and Fraser, 2000).

1.4 Outlook

Chapter 2 will introduce existing frameworks for single-agent and multiagent decision-theoretic planning. In the single-agent case we will discuss Markov decision processes (MDPs) and partially observable MDPs (POMDPs) and their associated solution concepts, value functions in particular. We will introduce value iteration as a standard method for computing optimal value functions, which encode optimal plans. For the cooperate multiagent case we will review the decentralized POMDP (DEC-POMDP) framework, which is a generalization of POMDPs to multiple agents.

In Chapter 3 we will present an overview of known techniques for solving POMDPs. We will discuss exact solution techniques and their poor scalability, motivating work on approximate POMDP solution methods. We will focus on a family of approximate algorithms known as point-based POMDP methods. Point-based approximate techniques for POMDPs compute a policy based on a finite set of belief points collected in advance from the agent's belief space. Next we will describe and discuss Perseus, our randomized point-based approximate POMDP planner. We will present experimental results in several benchmark domains and a robotic planning problem.

In Chapter 4 we will present methods to extend approximate POMDP planning in general (and Perseus in particular) to continuous domains. We will tackle planning for agents that have a continuous set of actions at their disposal and propose a technique for planning in continuous state spaces. We will discuss related work on handling continuous observation spaces, and ideas how to extend Perseus along these lines. We will conclude the chapter by highlighting several third-party extensions to Perseus.

In Chapter 5 we will switch to the multiagent setting, in which we will consider the problem of cooperative multiagent planning under uncertainty. We will consider the DEC-POMDP framework to compute plans for a team of agents that have the ability to communicate, but with limited bandwidth and at a certain cost. We will propose a decentralized model, in which each agent only reasons about its own local state and some uncontrollable state features, which are shared by all team members. In contrast to other approaches, we will model communication as an integral part of the agent's reasoning, in which the meaning of a message is directly encoded in the policy of the communicating agent.

Finally, in Chapter 6 we will present general conclusions, and outline promising directions of future research. We will also highlight application areas in which the methods developed in this thesis have been or could be applied.

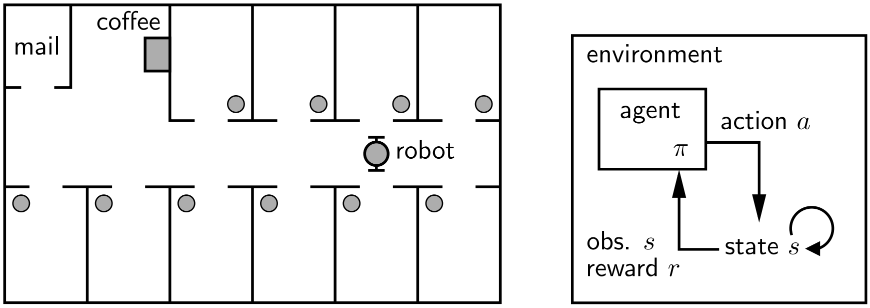
Chapter 2

An overview of decision-theoretic planning under uncertainty

In Chapter 1 we introduced the problem of planning under uncertainty, which is a key problem of intelligent agent design, one of the major goals of artificial intelligence. We will tackle uncertainty in the agent's environment by defining probabilistic models. The agent's environment reacts stochastically when the agent executes actions, and also the agent's sensor readings are related to the state of the environment in a probabilistic manner. We will focus on decision-theoretic planning, in which the goal of an agent is to compute a plan that maximizes expected utility. In this chapter we will provide an overview of decision-theoretic planning under uncertainty as an introduction for subsequent chapters. We will adopt extensions of Markov decision processes (MDPs, Sec. 2.2) as our planning framework, in particular partially observable MDPs (POMDPs, Sec. 2.3) and decentralized POMDPs (DEC-POMDPs, Sec. 2.4).

2.1 Introduction

In this thesis, we consider planning as the process of computing a (conditional) sequence of actions that fulfill a given task as well as possible. It is a crucial part of any intelligent agent; human, robot or software agent alike. We adopt Russell and Norvig (2003)'s definition that "an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators". As a running example in this chapter we will consider a service robot, which has been deployed in an office building to perform janitorial and other duties. The robot has to keep the building clean and in orderly condition, by cleaning the floors and emptying garbage cans. Furthermore, it



(a) Service-robot domain.

(b) MDP agent.

Figure 2.1: (a) An office environment with a service robot. The mail room is located in the top-left corner and next to it is the coffee lounge. Each office is equipped with a garbage can. (b) An agent in an MDP environment.

has to assist the office staff by delivering mail and making coffee. Fig. 2.1(a) shows an office environment in which the service robot should perform its duties.

Classic AI planning addresses the problem of finding a sequence of actions that takes the environment from a starting state to a goal state (Fikes and Nilsson, 1971; Korf, 1987). However, programming our robot to perform a fixed sequence of actions every day is not desirable, as the robot should respond to changes in the environment, e.g., execute a round of mail delivery after the mailman has stopped by. The robot should also be able to prioritize its duties, for instance enjoying a fresh pot of coffee is likely to be more important to its users than having an empty garbage can in their office. Such planning problems can be framed in (extensions of) the Markov decision process model, which originates from control theory and operations research (Bellman, 1957; Howard, 1960; Sondik, 1971), and has gained popularity in the AI community (Barto, Bradtke, and Singh, 1995; Sutton and Barto, 1998; Kaelbling et al., 1998; Boutilier et al., 1999; Russell and Norvig, 2003). First we will consider the problem of computing a plan for a single agent, followed by a characterization of the multiagent problem.

2.2 Markov decision processes

A Markov decision process (MDP) models the repeated interaction of an agent with a stochastic environment (Bellman, 1957; Puterman, 1994; Sutton and Barto, 1998; Boutilier et al., 1999; Bertsekas, 2000). The MDP framework is defined on the *state* of the system at hand, where the system consists of both the agent and the environment. The state is a description of the system at a certain point in time, and should contain all information relevant for the decision making. For example, in our service-robot domain the state should contain the current location of the robot, the status of the coffee pot and the garbage cans, whether or not there is mail waiting to be delivered, and how dirty each floor is. In general, the more detailed the state description is, the better the modeling of the real world will be, but a larger state space translates into a higher computational cost for planning. In an MDP the state description should at least contain enough information to have the Markov property, which means that an agent only has to consider its current state in order to act optimally. Given a Markovian state, an agent can forget about its history of states visited and actions executed, without adverse effects on its performance.

How the environment responds to the repeated interaction of the agent is modeled by state *transitions*. The state of the system can change every time the agent executes an action. For instance, if our robot chooses to clean a dirty floor, the state of the floor will most likely change from “dirty” to “clean”. We would like to model the possibility that the robot fails to clean the floor in a single cleaning action, as the floor might be dirtier than a single wipe can resolve. The uncertain outcome of executing actions is modeled in the MDP by a probabilistic transition model, which for instance specifies that after cleaning a floor it will actually be clean with probability 0.8, while there might still be some dirt left with probability 0.2. The state can also change due to external events, i.e., events outside of the agent’s control. External events can also be modeled in the transition model. For instance, the service robot has no influence on the arrival of the mailman, but there might be a 0.05 probability at every time step that the state of the mail room will switch from “no mail” to “mail waiting for delivery”, indicating that the mailman has stopped by.

The agent’s goal is to perform a task by executing a plan that fulfills the task best. We will call the expected performance of a plan its *value*, a concept which will play a central role in this thesis. In an MDP the agent’s task is defined by specifying the quality of executing an action in a particular state, which is modeled as a *reward* signal. Good actions receive positive reward, bad actions are punished with negative reward. In our service-robot domain for instance, making coffee when the coffee pot is empty could fetch the robot a reward of 10, while cleaning a clean floor only wastes energy and receives reward -1. By specifying different rewards for successfully fulfilling each duty the robot’s designer can prioritize the robot’s tasks. The performance of a particular plan is judged by how much reward it is expected to accumulate over time, i.e., the value of a plan is defined as a sum of all rewards the agent will gather during its lifetime.

We will now turn to a more formal treatment of the MDP model, which allows us to describe methods for computing optimal plans. Thorough reviews of the MDP framework are available (Boutilier et al., 1999; Bertsekas, 2000), while Sutton and Barto (1998) provide a gentle introduction.

2.2.1 Formal description

The MDP framework models stochastic environments in which an agent is uncertain about the exact effect of executing a certain action. This uncertainty is captured by a probabilistic transition model, which specifies the probabilistic effect of how each action changes the state. Throughout this thesis we will assume that time is discretized in time steps of equal length, and at the start of each step the agent has to execute an action. Continuous-time MDP settings have been mainly studied in the operations-research community (Puterman, 1994; Bertsekas, 2000). More formally, an MDP assumes that at any time step t the environment is in a state $s \in S$, the agent takes an action $a \in A$ and receives a deterministic reward $R(s, a)$ from the environment as a result of this action, while the environment switches to a new state s' according to a known stochastic transition model $p(s'|s, a)$. The Markov property entails that the next state s_{t+1} only depends on the previous state s_t and action a_t:

$$p(s_{t+1}|s_t, s_{t-1}, \ldots, s_0, a_t, a_{t-1}, \ldots, a_0) = p(s_{t+1}|s_t, a_t). \tag{2.1}$$

We will assume that the agent's initial state s_0 is drawn from a probability distribution over S. In this chapter we will assume that the state space S and the action space A are discrete and finite sets, but we will consider more general settings in Chapter 4. Fig. 2.1(b) shows the diagram of an MDP agent interacting with its environment.

The goal of the agent is to act in such a way as to maximize some form of long-term reward, i.e., a performance measure. As the problem is stochastic, we can only maximize the expected long-term reward, and we will consider the following performance measure:

$$E\Big[\sum_{t=0}^{h} \gamma^t R_t\Big], \tag{2.2}$$

where $E[\cdot]$ denotes the expectation operator, h is the planning horizon, and γ is a discount rate, $0 \leq \gamma < 1$. The discount rate ensures a finite sum when $h = \infty$ and is usually chosen close to 1. It captures the notion that a reward obtained in the near future is more valuable than a reward received in the far future. If it suffices for the agent to only consider a fixed number of steps in the future, i.e., when the agent's lifetime is limited, the horizon h can be set to a finite number, or to ∞ otherwise. For finite-horizon problems the discount rate is often discarded (i.e., $\gamma = 1$).

The behavior of the agent is defined by its plan, or policy in MDP terminology. A policy π instructs the agent what action it should take at every time step. A policy that specifies a fixed sequence of actions giving the initial state at $t = 0$ without checking the effects of the executed actions is called an open-loop plan, while a closed-loop plan conditions on the state of the agent at each time step t. In an MDP an open-loop plan is generally not sufficiently expressive as the transition model is stochastic. A closed-loop policy can mitigate the stochastic effects by considering the state of the agent after executing each action. An optimal policy π^* maximizes the performance measure, in our case (2.2). It is well-known that in this case a stationary and deterministic optimal policy exists for the infinite-horizon case (Howard, 1960; Puterman, 1994). The fact that we can restrict ourselves to non-randomized policies is attractive, as such policies are easier to evaluate, and we have to search a smaller policy space. A stationary policy is independent of the particular time step at which the agent is executing the policy, which also restricts the policy space. For the finite-horizon case, however, the particular time step is relevant.

Without loss of generality[1], the following discussion will focus on the $h = \infty$ case, and define an optimal policy $\pi^* : S \to A$, which for every state indicates the action that is optimal to execute, given that π^* will also be followed in the future. One way to characterize an MDP policy is to consider its value function $V^\pi : S \to \mathbb{R}$, which for every state s estimates the amount of discounted reward the agent can gather when it starts in s and acts according to π, which follows from (2.2):

$$V^\pi(s) = R(s, \pi(s)) + E\Big[\sum_{t=1}^{\infty} \gamma^t R(s_t, \pi(s_t))\Big]. \tag{2.3}$$

The expectation operator averages over the stochastic transition model, which leads to the following recursion, known as the Bellman recursion (1957):

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V^\pi(s'). \tag{2.4}$$

Many solution techniques focus on computing the optimal value function V^*, as we can easily extract the policy π corresponding to a particular value function V:

$$\pi(s) = \arg\max_{a \in A} \Big[R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V(s')\Big], \tag{2.5}$$

which instructs the agent to take the action which maximizes the sum of the immediate reward and the expected future discounted reward.

2.2.2 Dynamic programming

Bellman (1957) considered the problem of acting optimally in MDPs and introduced a family of techniques known as dynamic programming. The Bellman

[1] The time step can be encoded in the state description, discounting can be discarded by setting $\gamma = 1$, and if we also add an absorbing terminal state in which no reward can be obtained any more, we can convert any finite-horizon problem to an infinite-horizon one.

Algorithm 2.1 Value iteration.

```
Initialize $V$ arbitrarily, e.g., $V(s) = 0, \forall s \in S$
repeat
  $\delta \leftarrow 0$
  for all $s \in S$ do
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_{a \in A} \Big[R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)V(s')\Big]$
    $\delta \leftarrow \max(\delta, |v - V(s)|)$
until $\delta < \epsilon$
Return $V$
```

equation (2.4) can be turned into an optimality principle, which specifies the optimal value function

$$V^*(s) = \max_{a \in A} \Big[R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)V^*(s')\Big], \tag{2.6}$$

known as the Bellman equation. Solving this (nonlinear) systems of equations for each state s yields the optimal value function, and from it an optimal policy π^* using (2.5). However, due to the nonlinear max operator solving the system for each state simultaneously is not efficient for large MDPs (Puterman, 1994, Sec. 6.9), and Bellman introduced a successive approximation technique called *value iteration*. The optimal value function V_0^*, when the agent can only take one action, is defined by the reward model:

$$V_0^*(s) = \max_{a \in A} R(s,a). \tag{2.7}$$

In order to consider one step deeper into the future, i.e., to compute V_{n+1}^* from V_n^* we can turn (2.6) into an update:

$$V_{n+1}^*(s) = \max_{a \in A} \Big[R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)V_n^*(s')\Big], \tag{2.8}$$

which is a contraction mapping that will converge to the fixed point V^* (Puterman, 1994). This update operation is known as a Bellman *backup*, and we denote the operator as H_{MDP}, allowing us to write (2.8) as

$$V_{n+1}^* = H_{\text{MDP}} V_n^*. \tag{2.9}$$

Algorithm 2.1 shows pseudocode for the plain value-iteration algorithm, in which sweeps are made over the full state space, known as exhaustive backups, backing up every state in turn until convergence (Sutton and Barto, 1998). Value iteration has converged when the value function stabilizes, i.e., when the largest update δ in an iteration is below a certain threshold ϵ. Value iteration still converges, however, when instead of exhaustive backups arbitrary states are backed up in arbitrary order, provided that in the limit all states are visited infinitely often (Bertsekas and Tsitsiklis, 1989). We can take advantage of this flexibility in order to speed up the algorithm, for instance by backing up the most promising states first, a technique known as prioritized sweeping (Moore and Atkeson, 1993; Peng and Williams, 1993).

Exact dynamic-programming approaches require full knowledge of the system's transition and reward models. If the agent does not possess such knowledge, we can still perform approximate sample-based backups using so-called reinforcement-learning techniques.

2.2.3 Reinforcement learning

Model-free techniques for sequential decision making focus on learning to act optimally in unknown environments, and are collectively known as reinforcement learning (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998). A policy is learned by experiencing the environment over and over again. The experience could for instance be used to approximate the unknown MDP model, which then can be solved by model-based dynamic-programming techniques. A more common approach, however, is to directly learn a policy. The intuition is that for learning a successful policy one might not need to have detailed and accurate knowledge of the underlying MDP model.

Reinforcement learning often considers learning a policy in the form of a $Q : S \times A \to \mathbb{R}$ function, which estimates the value of executing action a in state s, and the policy defined by a Q-function is simply $\pi(s) = \arg\max_{a \in A} Q(s, a)$. A commonly used method to learn an optimal Q-function is Q-learning (Watkins, 1989). This starts with an arbitrarily initialized Q-function, and in order to improve on this policy, the agent executes the policy and uses the experience to update the Q-values. In particular, every time the agent takes action a in state s, transitions to state s' and receives reward $R(s, a)$ it updates its Q-function as follows:

$$Q(s,a) = (1-\beta)\; Q(s,a) + \beta\Big[R(s,a) + \gamma \max_{a' \in A} Q(s',a')\Big], \tag{2.10}$$

where $0 < \beta \leq 1$ is a learning rate. It can be interpreted as adjusting the $Q(s,a)$ estimate in the direction of the immediate reward $R(s,a)$ and the discounted estimated value of the successor state s'. Q-learning is guaranteed to converge to the optimal Q-values and therefore to π^* if all $Q(s,a)$ values are updated infinitely often (Watkins and Dayan, 1992), at which point it will satisfy the Bellman equation (2.6). In order to make sure all actions will eventually be tried in all states exploration is necessary. A common exploration method is to execute a random action instead of the one prescribed by the Q-function with small probability ϵ, which is known as ϵ-greedy exploration (Sutton and Barto, 1998).

2.2.4 Factorized state representations

In the previous discussion of MDPs we have considered a state space S that consists of a discrete and finite set of states, without assuming any structure. Returning to our service-robot example, we can see that its state description is the combination of a number of state features: the location of the robot, and the status of the coffee pot, garbage cans, floors, etc. As such, one can also view the state of the system as the cross-product of all these k features, and represent the state in a factorized state space $S = S_1 \times S_2 \times \cdots \times S_k$ (Boutilier, Dearden, and Goldszmidt, 2000; Guestrin, Koller, Parr, and Venkataraman, 2003). A factorized state space allows one to compactly represent the transition function as a dynamic Bayesian network (Jensen, 2001; Murphy, 2002), and exploits (assumed) independence relationships between features. For instance, we can safely assume that the status of the coffee pot at time step $t+1$ is independent of the amount of garbage deposited in a particular garbage can at time t. Apart from ease of representation, such a compactly represented transition model can be exploited when solving a factored MDP (Hoey, St-Aubin, Hu, and Boutilier, 1999; Guestrin et al., 2003).

2.3 Partially observable Markov decision processes

One of the limiting factors of the MDP model is its assumption that the agent knows with full certainty the true state of the environment at all times, a strong assumption that can restrict the applicability of the framework. In our service-robot domain, for instance, the robot is unlikely to be able to tell whether any of the garbage cans is full or empty unless it is located near to it. Whether or not there is mail is waiting to be delivered is only detectable inside the mail room. When the robot has left the mail room, the mailman might drop by and deliver mail (without the robot noticing it), and as such, the robot is uncertain about the status of the mail room, until it visits the room again. Another source of uncertainty regarding the true state of the system are imperfections in the robot's sensors. For instance, let us suppose the robot uses a camera to check whether a floor is dirty or not. The vision algorithm processing the camera images is likely to make mistakes sometimes, reporting a clean floor as dirty or vice versa. Such an imperfect sensor also prevents the robot from knowing the true state of the system: even if the vision algorithm reports a clean floor, it is still possible that the floor is actually dirty.

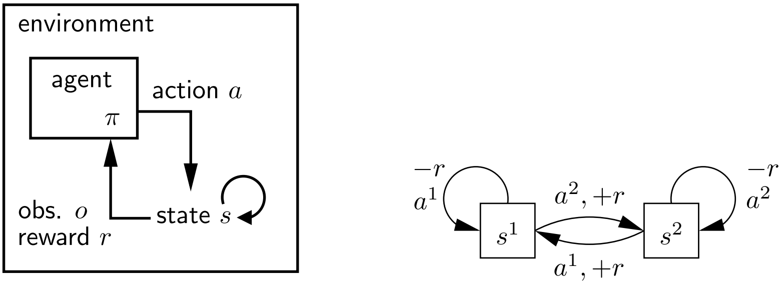
Extending the MDP setting, a partially observable Markov decision process (POMDP) also deals with uncertainty resulting from the agent's imperfect sensors (Stratonovich, 1960; Drake, 1962; Dynkin, 1965; Aoki, 1965; Åström, 1965; Sondik, 1971; Lovejoy, 1991; Kaelbling et al., 1998). It allows for planning in environments which are only partially observable to the agent, i.e., environments in which the agent cannot determine with full certainty the true state of the environment. In general the partial observability stems from two sources: (1) multiple states give the same sensor reading, in case the agent can only sense a limited part of the environment, and (2) its sensor readings are noisy: observing the same state can result in different sensor readings. The partial observability can lead to "perceptual aliasing": different parts of the environment appear similar to the agent's sensor system, but require different actions. The POMDP captures the partial observability by a probabilistic observation model, which relates possible observations to states.

2.3.1 Formal description

As in the fully observable MDP model, the agent takes an action a in state s, the environment transitions to state s' according to $p(s'|s,a)$ and the agent receives an immediate reward $R(s,a)$. The agent then perceives an observation $o \in O$, that may be conditional on its action, which provides information about the state s' through a known stochastic observation model $p(o|s,a)$. We will assume throughout this thesis that the observation space O is discrete and finite, but in Sec. 4.4.3 we will discuss related work that extends POMDP planning to continuous observation spaces. Fig. 2.2(a) illustrates these concepts by depicting a schematic representation of a POMDP agent.

2.3.2 Memory

In order for an agent to choose its actions successfully in partially observable environments some form of memory is needed, as the observations the agent receives do not provide an unique identification of s. For instance, consider the two-state infinite-horizon POMDP depicted in Fig. 2.2(b), presented by Singh, Jaakkola, and Jordan (1994). The agent has two actions, one of which will deterministically transport it to the other state, while executing the other action has no effect on the state. If the agent jumps to the other state receives a reward of $r > 0$, or $-r$ otherwise. The optimal policy in the underlying MDP has a value of $\frac{r}{1-\gamma}$, as the agent can gather a reward of r at each time step. In the POMDP however, the agent receives the same observation in both states. As a result, there are only two memoryless deterministic stationary policies possible: always execute a^1 or always execute a^2. The maximum expected reward of these policies is $r - \frac{\gamma r}{1-\gamma}$, when the agent successfully jumps to the other state at the first time step. If we allow stochastic policies, the best stationary policy would



(a) A POMDP agent.

(b) Two-state POMDP.

Figure 2.2: (a) A POMDP agent interacting with its environment. (b) A two-state POMDP from (Singh et al., 1994), in which the agent receives the same observation in both states.

yield an expected discounted reward of 0, when it chooses either action 50% of the time. However, if the agent could remember what actions it had executed, it could execute a policy that alternates between executing a^1 and a^2. Such a memory-based policy would gather $\frac{\gamma r}{1-\gamma} - r$ in the worst case, which is close to the optimal value in the underlying MDP (Singh et al., 1994).

2.3.3 Belief states

The example in the previous section illustrates the need for memory when planning in a POMDP. A straightforward implementation of memory would be to simply store the sequence of actions executed and observations received. However, such a form of memory can grow indefinitely over time, turning it impractical for large or infinite-horizon problems. Fortunately, a better option exists, as we can transform the POMDP to a belief-state MDP in which the agent summarizes all information about its past using a belief vector $b(s)$ (Stratonovich, 1960; Dynkin, 1965; Åström, 1965). The belief b is a probability distribution over S, which forms a Markovian signal for the planning task. The belief is a sufficient statistic of the history, which means we could not do any better even if we had remembered the full history of actions and observations. All beliefs are contained in a $(|S|-1)$-dimensional simplex $\Delta(S)$, hence we can represent a belief using $|S|-1$ numbers. Each POMDP problem assumes an initial belief b^0, which for instance can be set to a uniform distribution over all states (representing complete ignorance regarding the initial state of the environment). Every time the agent takes an action a and observes o, its belief is updated by Bayes' rule:

$$b^{ao}(s') = \frac{p(o|s',a)}{p(o|b,a)} \sum_{s \in S} p(s'|s,a) b(s), \tag{2.11}$$

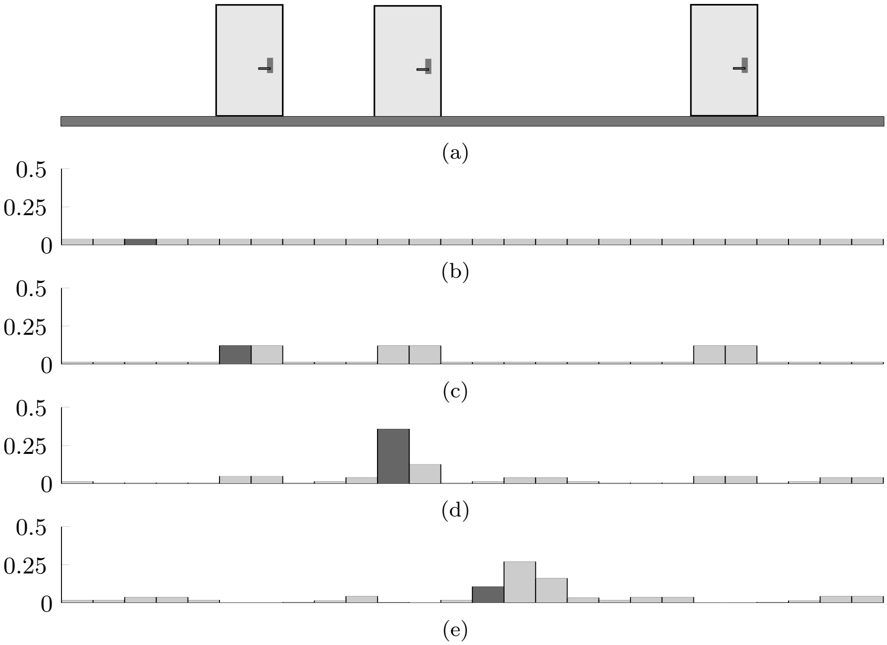


Figure 2.3: Belief-update example (adapted from Fox et al., 1999). (a) A robot moves in a one-dimensional corridor with three identical doors. (b)-(e) The evolution of the belief over time, for details see main text.

where $p(o|b,a) = \sum_{s' \in S} p(o|s',a) \sum_{s \in S} p(s'|s,a)b(s)$ is a normalizing constant.

Fig. 2.3 shows an example of a sequence of belief updates for a robot navigating in a corridor with three identical doors. The corridor is discretized in 26 states and is circular, i.e., the right end of the corridor is connected to the left end. The robot can observe either *door* or *corridor*, but its sensors are noisy. When the robot is positioned in front of a door, it observes *door* with probability 0.9 (and *corridor* with probability 0.1). When the robot is not located in front of a door the probability of observing *corridor* is 0.9. The robot has two actions, *forward* and *backward* (right resp. left in the figure), which transport the robot 3 (20%), 4 (60%), or 5 (20%) states in the corresponding direction. The initial belief b^0 is uniform, as displayed in Fig. 2.3(b). Fig. 2.3(c) through (e) show how the belief of the robot is updated as it executes the *forward* action each time. The true location of the robot is indicated by the dark-gray component of its belief. In Fig. 2.3(c) we see that robot is located in front of the first door, and although it is fairly certain it is located in front of a door, it cannot tell which one. However, after taking another move forward it again observes *door*, and now can pinpoint its location more accurately, because of the particular configuration of the three doors (Fig. 2.3(d)). However, in Fig. 2.3(e) the belief blurs again, which is due to the noisy transition model and the fact that the *corridor* observation is not very informative in this case.

2.3.4 Value functions

As in the fully observable MDP setting, the goal of the agent is to choose actions which fulfill its task as well as possible, i.e., to compute an optimal plan. Such a plan is called a policy $\pi(b)$ and maps beliefs to actions. Note that, contrary to MDPs, the policy $\pi(b)$ is a function over a continuous set of probability distributions over S. A policy π can be characterized by a value function $V^\pi : \Delta(S) \to \mathbb{R}$ which is defined as the expected future discounted reward $V^\pi(b)$ the agent can gather by following π starting from belief b:

$$V^\pi(b) = E_\pi\Big[\sum_{t=0}^{h} \gamma^t R(b_t, \pi(b_t))\Big| b_0 = b\Big], \tag{2.12}$$

where $R(b_t, \pi(b_t)) = \sum_{s\in S} R(s, \pi(b_t))b_t(s)$, and h is the planning horizon.

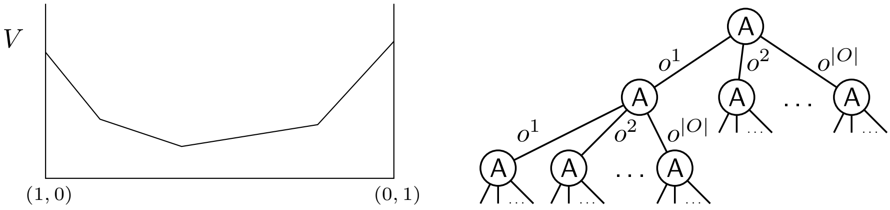
A policy π which maximizes V^π is called an optimal policy π^*; it specifies for each b the optimal action to execute at the current step, assuming the agent will also act optimally at future time steps. The value of an optimal policy π^* is defined by the optimal value function V^*, that satisfies the Bellman optimality equation $V^* = H_{\mathrm{POMDP}}V^*$:

$$V^*(b) = \max_{a\in A}\Big[\sum_{s\in S} R(s,a)b(s) + \gamma \sum_{o\in O} p(o|b,a)V^*(b^{ao})\Big], \tag{2.13}$$

with b^{ao} given by (2.11). When (2.13) holds for every $b \in \Delta(S)$ we are ensured the solution is optimal.

Computing value functions over a continuous belief space might seem intractable at first, but fortunately the value function has a particular structure that we can exploit (Sondik, 1971). It can be parameterized by a finite number of vectors and has a convex shape. The convexity implies that the value of a belief close to one of the corners of the belief simplex $\Delta(S)$ will be high. In general, the less uncertainty the agent has over its true state, the better it can predict the future, and as such take better decisions. A belief located exactly at a particular corner of $\Delta(S)$, i.e., $b(s) = 1$ for a particular s, defines with full certainty the state of the agent. In this way, the convex shape of V can be intuitively explained. An example of a convex value function for a two-state POMDP is shown in Fig. 2.4(a). As the belief space is a simplex, we can represent any belief in a two-state POMDP on a line, as $b(s^2) = 1 - b(s^1)$. The corners of the belief simplex are denoted by $(1, 0)$ and $(0, 1)$, which have a higher value than a belief in center of the belief space, e.g., $(0.5, 0.5)$.

An alternative way to represent policies in POMDPs is by considering *policy trees* (Kaelbling et al., 1998). Fig. 2.4(b) shows a partial policy tree, in which



(a) Example value function.

(b) Example policy tree.

Figure 2.4: (a) An example of a value function in a two-state POMDP. The y-axis shows the value of each belief, and the x-axis depicts the belief space $\Delta(S)$, ranging from $(1, 0)$ to $(0, 1)$. (b) An example policy tree, where at node the agent takes an action, and it transitions to a next node based on the received observation $o \in \{o^1, o^2, \ldots, o^{|O|}\}$.

the agent starts at the root node of tree. Each node specifies an action which the agent executes at the particular node. Next it receives an observation $o \in \{o^1, o^2, \ldots, o^{|O|}\}$, which determines to what next node the agent transitions. The depth of the tree depends on the planning horizon h, i.e., if we want the agent to consider taking h steps, the corresponding policy tree has depth h. Without going into details, from such a policy tree we can derive a set of vectors that implement its policy: each vector represents the value of a particular node in the tree. Certain POMDP solution algorithms search in policy trees (Satia and Lave, 1973; Hansen, 1998a; Smith and Simmons, 2004), but we will focus on representing value functions as sets of vectors.

Computing value functions for POMDPs is a major focus of this thesis, and we will devote Chapters 3 and 4 to it. Next we will turn to the third source of uncertainty that we will be considering: the behavior of teammates.

2.4 Decentralized partially observable Markov decision processes

Until sofar we have concerned ourselves with models for computing plans involving a single agent. However, in our service-robot example multiple robots might be deployed in the same office environment. In this case we would like the robots to cooperate, instead of acting independently which clearly is suboptimal. For instance, multiple robots might be trying to make coffee at the same time. Instead, a better allocation of resources might be for one of them to make coffee while the others wipe floors or deliver mail. However, in the partially observable environments that we are considering, planning for multiple agents becomes significantly harder, even when they cooperate. We will first discuss two multiagent MDP extensions, a centralized and a decentralized one.

2.4.1 Multiagent MDPs

The MDP framework traditionally deals with planning problems in which a single agent is involved, but it can be extended to a cooperative multiagent setting in a straightforward manner. Boutilier (1996) introduced the *multiagent MDP* (MMDP), which considers a set $I = \{1, \ldots, m\}$ of m agents. Each agent i has its own action set A_i, but the transitions of the system are governed by $p(s'|s, \bar{a})$, in which $\bar{a} \in A_1 \times \cdots \times A_m$ is the joint action of all agents combined. The agents act simultaneously and receive the same reward $R : S \times A_1 \times \cdots \times A_m \to \mathbb{R}$. In this way, the problem of computing an optimal policy for the team of agents is reduced to planning optimally for a single agent with a large number of available actions. Such a joint MDP can be readily solved by dynamic-programming techniques (Sec. 2.2.2), but will be infeasible for large teams, as the size of the joint action set grows exponentially in the number of agents.

The MMDP model assumes that each agent in the team can observe the true global state of the system at each time step. However, in such a distributed setting it might be more natural to assume that each agent observes its local state, and that the global state of the system is composed of the local states of all agents. Becker, Zilberstein, Lesser, and Goldman (2004) study a decentralized setting, known as a *decentralized MDP* (DEC-MDP), in which the system state space is factored into $m+1$ components, $S = S_0 \times S_1 \times \cdots \times S_m$. The S_0 component of the state refers to uncontrollable shared state features, which each agent can observe and are relevant for the team's decision making, but are outside the control of the agents. A good example is the time of the decision process, which is relevant in finite-horizon tasks or when precisely timed coordination is required. The S_i components (where $1 \leq i \leq m$) refer to the state of each agent individually, for instance its location in the world. Each agent bases its decision making on its local state $\bar{s}_i \in S_0 \times S_i$ only. In general it will be infeasible to compute a globally optimal solution based on local state information only, but Becker et al. (2004) present a technique for computing optimal policies in decentralized MDPs, assuming transition independence between agents and a special joint reward structure.

2.4.2 Formal description of decentralized POMDPs

As in the fully observable MDP case, POMDPs have also been extended to cooperative multiagent settings. The decentralized POMDP (DEC-POMDP) model is one such straightforward extension (Bernstein et al., 2002). It is essentially equivalent to the multiagent team decision problem framework (Pynadath and Tambe, 2002) or the partially observable identical payoff stochastic game (Peshkin, Kim, Meuleau, and Kaelbling, 2000). If the agents are self-interested, i.e., have different reward functions, the model is known as a partially observable stochastic game (POSG) (Hansen, Bernstein, and Zilberstein, 2004).

The DEC-POMDP model considers a set $I = \{1, \ldots, m\}$ of m agents, which share the same reward function. Each agent i has its own action set A_i, observation set O_i, and initial belief $b_i^0 \in \Delta(S)$. The joint transition model $p(s'|s, \bar{a})$ defines the probability of jumping to state $s' \in S$ given that the team executed joint action $\bar{a} \in A_1 \times \cdots \times A_m$ in $s \in S$. The joint observation model $p(\bar{o}|s, \bar{a})$ defines the probability that after taking joint action $\bar{a}$ and ending up in state s results in joint observation $\bar{o} \in O_1 \times \cdots \times O_m$. The reward function $R : S \times A_1 \times \cdots \times A_m \to \mathbb{R}$ gives a scalar reward signal to the team. Fig. 2.5 shows agents i and j interacting with their environment.

2.4.3 Solving decentralized POMDPs

Contrary to the single-agent case, in a DEC-POMDP it is not possible in general to compute a belief state as an agent only knows its local observation o_i but not the complete observation vector $\bar{o}$, which is required for computing the belief update (2.11). As such, each agent chooses its action a_i at time t based on its policy $\pi_i : \times_{t-1}(A_i \times O_i) \to A_i$, i.e., it only considers its own history of actions taken and observations received until time t. The goal of the team of agents is to compute a joint policy $\pi = \{\pi_1, \ldots, \pi_m\}$ which maximizes the future discounted reward $E_\pi\left[\sum_{t=0}^{h} \gamma^t R_t\right]$, as in the MDP and POMDP case. As the agents form a team, we assume all static models (including the initial state distribution) are common knowledge among all agents, as well as each of their individual policies. Equivalently, we can assume that given the models each agent can solve independently and in parallel the problem at hand, in which case the resulting policies will also be common knowledge (Xuan, Lesser, and Zilberstein, 2001; Kok, Spaan, and Vlassis, 2005).

As solving a DEC-POMDP in a general setting is highly intractable (NEXP-complete, Bernstein et al., 2002), most research has been devoted to studying simplified settings, in which additional assumptions are made on the environment. We will now discuss three possible angles for simplifying DEC-POMDPs: observability assumptions, the level of centralization, and communication aspects.

2.4.4 Observability

In a team of agents each agent will carry its own sensors, and as such will receive a (possibly noisy) sensor reading related to the part of the state space it can observe. A common assumption is that each agent's observations are independent from those of its teammates (Nair, Tambe, Yokoo, Pynadath, and Marsella, 2003; Goldman and Zilberstein, 2003; Roth, 2005). In a standard POMDP setting the agent implicitly observes its own action (as it is used to

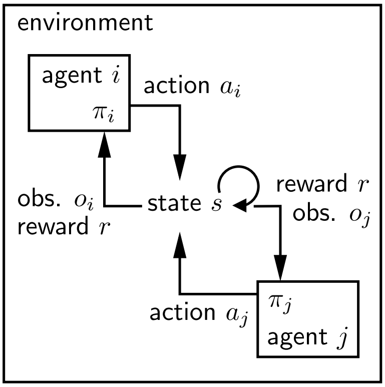


Figure 2.5: Two agents in a DEC-POMDP.

update its belief), but in a multiagent setting an agent cannot observe the actions chosen by its teammates, and so the agent does not know the executed joint action. Given only its own observation and action it cannot update the belief of the complete team, but it can reason about the possible joint beliefs of the team (Emery-Montemerlo, Gordon, Schneider, and Thrun, 2004; Roth, 2005; Roth, Simmons, and Veloso, 2005).

Another simplifying assumption that can be made is local full observability: the global state can be factored in a number of components and each agent observes a local state component with full certainty, which turns the model into a decentralized MDP (DEC-MDP) (Xuan et al., 2001; Becker et al., 2004). An advantage of local full observability is the fact that only a finite amount of policies have to be considered, as there are only a finite number of possible deterministic mappings from states to actions (Goldman and Zilberstein, 2003; Becker et al., 2004).

2.4.5 Level of centralization

If a team of agents is allowed to communicate for free and without limitations, each agent can broadcast at every time step its perceived observation to all of its teammates. As a result every agent knows the joint observation vector which it uses to update the joint belief of the team, based on which the next joint action is selected. As such, free communication reduces the distributed control problem to a centralized, single-agent one, which can be solved using standard POMDP solution techniques. However, in reality communication is not free or unlimited, and modeling a team of agents in this way is not desirable.

Nevertheless, a common approach is to treat the multiagent system as if communication were free, solve the centralized POMDP, and execute the resulting joint policy in a distributed fashion while imposing communication constraints (Goldman and Zilberstein, 2003; Emery-Montemerlo et al., 2004; Roth, 2005; Roth et al., 2005). The assumption is that the centralized POMDP can be (approximately) solved, and the focus is on distributed execution of the team's policy instead of computing it in a distributed manner. While executing the policy, a limited form of communication is used to attempt to preserve coherent behavior of the team. Note that the communication decisions are not part of the centralized policy, but are made by a separate algorithm which monitors the uncertainty regarding the joint belief.

Xuan et al. (2001) propose a model that incorporates the communication decision directly in the agent's policy. It adds a communication sub-stage to the decision process, in which an agent decides whether it will communicate with its teammates. In this way an agent can explicitly reason about communication, instead of relying on an independent instrument to handle communication issues. Extending this framework, Goldman and Zilberstein (2003) present a formal model for decentralized control with communication decisions based on the DEC-POMDP model (Bernstein et al., 2002).

2.4.6 Communication

The communication abilities of each agent can be classified in three categories: (1) free communication, when an unlimited amount of messages can be send to teammates at zero cost, (2) no communication, when no messages can be exchanged between teammates or the cost to do so is prohibitively high, and (3) general communication, the case when an agent has the capability to communicate but sending messages comes at a certain cost (Roth, 2005). The communication cost is generally modeled as a negative reward signal.

We will focus on the general communication case, which requires deciding when to communicate, what the message should contain and whom the message should be sent to. Addressing the last question, two types of communication are commonly used, broadcast and peer-to-peer. The latter entails sending a message from one agent to a single other agent, while broadcasting a message relays the message to all agents in the team. Evidently, broadcast communication can be expensive, but many distributed POMDP methods rely on it during policy execution to ensure that all team members have a synchronized model of the joint belief.

The contents of a message can be arbitrary, but a common approach is to communicate (part of the) action-observation history of an agent, as this reduces the uncertainty in the tree of possible joint beliefs. In general a message is sent when the agent considers the expected gain in future reward to outweigh the cost of communicating. Most studies assume that communication is instantaneous and reliable, indicating that every message that is sent out arrives immediately at its destination and its contents remains unaltered.

Model	Finite horizon	Infinite horizon
MDP	P-complete	P-complete
POMDP	PSPACE-complete	Undecidable
DEC-POMDP	NEXP-complete	Undecidable

Table 2.1: Complexity results for solving various MDP models. The infinite-horizon results refer to the total discounted reward case, and the finite-horizon results assume that the horizon is lower than the number of states. Sources: MDP and POMDP finite horizon (Papadimitriou and Tsitsiklis, 1987), POMDP infinite horizon (Madani et al., 2003), and DEC-POMDP (Bernstein et al., 2002).

2.5 Discussion

In this chapter we provided an overview of decision-theoretic planning models, cast in the Markov decision process framework. POMDPs allow for dealing with uncertainty in actuators and sensors, extending classic AI planning. We also discussed a multiagent extension of POMDPs, known as DEC-POMDPs, in which planning is performed for a team of agents rather than for a single agent. We focused on the models and basic solution techniques for MDPs, postponing the discussion of (DEC-)POMDP solution methods to subsequent chapters.

To relate the cost of solving these models, Table 2.1 compares the best known complexity results (Papadimitriou, 1993) for the three models we have discussed. We see that MDPs are solvable in polynomial time, thus allowing for efficient implementation (Papadimitriou and Tsitsiklis, 1987). Adding partial observability to the problem dramatically increases its complexity, as for the infinite-horizon case it is impossible to tell whether a given discounted reward can be achieved in a particular POMDP (Madani, Hanks, and Condon, 2003), and the finite-horizon complexity is PSPACE-complete indicating there is no polynomial-time algorithm for solving the POMDP (unless P = PSPACE) (Papadimitriou and Tsitsiklis, 1987). Moving to multiple agents causes an exponential jump in complexity, even for the case of only two agents (Bernstein et al., 2002), where there is provably no polynomial algorithm for solving the DEC-POMDP. Quoting Madani et al. (2003), the "discovery that interesting problems are computationally intractable in the worst case should neither be surprising nor discouraging to the AI researcher".

In the following chapters we will use the decision-theoretic planning frameworks presented in this chapter to develop approximate but tractable planning algorithms. In Chapter 3 we will review exact and approximate algorithms for solving POMDPs, and present Perseus, our randomized approximate POMDP solver. In Chapter 4 we will extend approximate POMDP planning in general and Perseus in particular to continuous action and state spaces. Chapter 5 will switch to the cooperative multiagent setting, in which we will use the DEC-POMDP framework for planning for teams of communicating agents.

Chapter 3

Planning in partially observable stochastic environments

The previous chapter introduced models and solution concepts for planning under various kinds of uncertainty, both for single agents as well as multiagent teams. In this chapter we will focus exclusively on planning for single agents, using the partially observable Markov decision process (POMDP) framework. First we will review exact value-iteration methods for solving POMDPs, followed by describing a family of approximate POMDP algorithms, known as the point-based methods. The main contribution of this chapter is our randomized point-based POMDP solver called PERSEUS which we will present in Sec. 3.6. After a review of related work we will present competitive results applying PERSEUS on benchmark domains as well as results from an office delivery task involving a mobile robot with omnidirectional vision in a highly perceptually aliased office environment.

3.1 Introduction

A major goal of Artificial Intelligence is to build intelligent agents (Russell and Norvig, 2003). An intelligent agent, whether physical or simulated, should be able to autonomously perform a given task, and is often characterized by its sense–think–act loop: it uses sensors to observe the environment, considers this information to decide what to do, and executes the chosen action. The agent influences its environment by acting and can detect the effect of its actions by sensing: the environment closes the loop. We are interested in computing a *plan* that maps sensory input to the optimal action to execute a given task. We consider types of domains in which an agent is uncertain about the exact consequence of its actions. Furthermore, it cannot determine with full certainty the state of the environment with a single sensor reading, i.e., the environment is only partially observable to the agent.

Planning under these kinds of uncertainty is a challenging problem as it requires reasoning over all possible futures given all possible histories. As we detailed in Chapter 2, partially observable Markov decision processes (POMDPs) provide a rich mathematical framework for acting optimally in such partially observable and stochastic environments (Stratonovich, 1960; Dynkin, 1965; Aoki, 1965; Åström, 1965; Sondik, 1971; Lovejoy, 1991; Kaelbling et al., 1998). The POMDP defines a sensor model specifying the probability of observing a particular sensor reading in a specific state and a stochastic transition model which captures the uncertain outcome of executing an action. The agent's task is represented by the reward it receives at each time step and its goal is to maximize the discounted cumulative reward. Assuming discrete models, the POMDP framework allows for capturing all uncertainty introduced by the transition and observation model by defining and operating on the *belief state* of an agent. A belief state is a probability distribution over all states and summarizes all information regarding the past.

The use of belief states allows one to transform the original discrete-state POMDP into a continuous-state Markov decision process (MDP). Recall that we can represent a plan in an MDP by its value function, which for every state estimates the amount of discounted reward the agent can gather when it acts according to the particular plan. In a POMDP the optimal value function, i.e., the value function corresponding to an optimal plan, exhibits particular structure (it is piecewise linear and convex) that one can exploit in order to facilitate the solving. Value iteration, for instance, is a method for solving POMDPs that builds a sequence of value-function estimates which converge to the optimal value function for the current task (Sondik, 1971). A value function in a finite-horizon POMDP is parameterized by a finite number of hyperplanes, or vectors, over the belief space, which partition the belief space in a finite amount of regions. Each vector maximizes the value function in a certain region and has an action associated with it, which is the optimal action to take for beliefs in its region. As we will explain next, computing the next value-function estimate—looking one step deeper into the future—requires taking into account all possible actions the agent can take and all subsequent observations it may receive. Unfortunately, this leads to an exponential growth of vectors with the planning horizon. Many of the computed vectors will be useless in the sense that their maximizing region is empty, but identifying and subsequently pruning them is an expensive operation.

Exact value-iteration algorithms (Sondik, 1971; Cheng, 1988; Cassandra, Kaelbling, and Littman, 1994) search in each value-iteration step the complete belief simplex for a minimal set of belief points that generate the necessary set of vectors for the next-horizon value function. This typically requires linear programming and is therefore costly in high dimensions. Other exact value-iteration algorithms focus on generating all possible next-horizon vectors followed by or interleaved with pruning dominated vectors in a smart way (Monahan, 1982; Zhang and Liu, 1996; Cassandra, Littman, and Zhang, 1997; Feng and Zilberstein, 2004; Lin, Bean, and White, 2004; Varakantham, Maheswaran, and Tambe, 2005). However, pruning again requires linear programming. Zhang and Zhang (2001) argued that value iteration still converges to the optimal value function if exact value-iteration steps are interleaved with approximate value-iteration steps in which the new value function is an upper bound to the previously computed value function. This results in a speedup of the total algorithm, however, linear programming is again needed in order to ensure that the new value function is an upper bound to the previous one over the complete belief simplex. In general, computing exact solutions for POMDPs is an intractable problem (Sec. 2.5; Papadimitriou and Tsitsiklis, 1987; Madani et al., 2003), calling for approximate solution techniques (Lovejoy, 1991; Hauskrecht, 2000).

In practical tasks one would like to compute solutions only for those parts of the belief simplex that are reachable, i.e., that can be actually encountered by interacting with the environment. This has recently motivated the use of approximate solution techniques which focus on the use of a sampled set of *belief points* on which planning is performed (Hauskrecht, 2000; Poon, 2001; Roy and Gordon, 2003; Pineau, Gordon, and Thrun, 2003a; Spaan and Vlassis, 2005a), a possibility already mentioned by Lovejoy (1991). The idea is that instead of planning over the complete belief space of the agent (which is intractable for large state spaces), planning is carried out only on a limited set of prototype beliefs that have been sampled by letting the agent interact (randomly) with the environment. PBVI, for instance, builds successive estimates of the value function by updating the value and its gradient only at the points of a (dynamically growing) belief set (Pineau et al., 2003a).

3.2 Value functions

Computing an optimal plan for an agent means solving the POMDP, and a classical method is value iteration (Sec. 2.2.2; Puterman, 1994). We transform the POMDP into an MDP defined over belief states, and the agent's policy $\pi(b)$ maps beliefs to actions. Recall that a policy π can be characterized by a value function $V^\pi : \Delta(S) \to \mathbb{R}$ which is defined as the expected future discounted reward $V^\pi(b)$ the agent can gather by following π starting from belief b:

$$V^\pi(b) = E_\pi\Big[\sum_{t=0}^{h} \gamma^t R(b_t, \pi(b_t))\Big| b_0 = b\Big]. \tag{2.12}$$

The value of an optimal policy π^* is defined by the optimal value function V^* which we compute by iterating a number of stages, at each stage considering a step further into the future. At each stage we apply the exact dynamic-programming operator H_{POMDP}, or some approximate operator $\tilde{H}$. To simplify notation we use H instead of H_{POMDP} to refer to the dynamic-programming operator for POMDPs, and we repeat it here for convenience:

$$H : V^*(b) = \max_{a \in A} \Big[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{o \in O} p(o|b,a) V^*(b^{ao}) \Big], \tag{2.13}$$

where b^{ao} is the updated belief, which is the result of receiving observation o after taking action a in belief b. It is computed by Bayes' rule (see Sec. 2.3.3):

$$b^{ao}(s') = \frac{p(o|s',a)}{p(o|b,a)} \sum_{s \in S} p(s'|s,a) b(s). \tag{2.11}$$

If the agent has only one time step left to act, we only have to consider the immediate reward for the particular belief b, and can ignore any future value $V^*(b^{ao})$ and (2.13) reduces to:

$$V_0^*(b) = \max_a \Big[\sum_s R(s,a) b(s) \Big]. \tag{3.1}$$

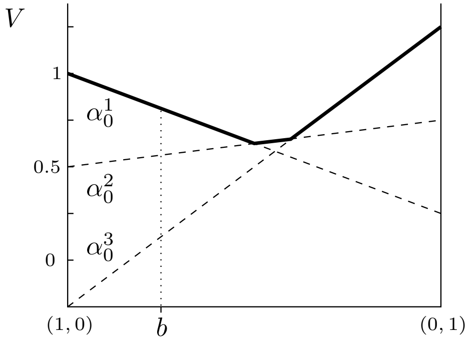
We can view the immediate reward function $R(s,a)$ as a set of $|A|$ vectors $\alpha_0^a = (\alpha_0^a(1), \ldots, \alpha_0^a(|S|))$, one for each action a:

$$\alpha_0^a(s) = R(s,a). \tag{3.2}$$

Now we can rewrite (3.1) as follows, where we view b as an $|S|$-dimensional vector:

$$\begin{aligned} V_0^*(b) &= \max_a \sum_s \alpha_0^a(s) b(s), && (3.3) \\ &= \max_{\{\alpha_0^a\}_a} b \cdot \alpha_0^a, && (3.4) \end{aligned}$$

where $(\cdot)$ denotes inner product. In Fig. 3.1(a) we plotted an example V_0^* value function for a POMDP with two states and three actions, and whose reward model is defined in Fig. 3.1(b). It illustrates how we can we view the immediate reward $R(s,a)$ as a set of vectors $\{\alpha_0^a\}$. In order to compute the value of example belief $b = (0.75, 0.25)$, we compute its inner product with all three vectors and maximize, resulting in $V_0^*(b) = 0.8125$. In order to know what the optimal action is at b, we consider the action associated with the maximizing vector, which in this case is a_1. However, this example only considers the immediate reward while ignoring reward the agent might gather in the future. In the



(a) Immediate-reward value function.

$R(s, a)$	a_1	a_2	a_3
s_1	1.00	0.50	−0.25
s_2	0.25	0.75	1.25

(b) Reward model.

Figure 3.1: (a) The value function V_0^* for a particular two-state POMDP with three actions, as induced by its reward model (b). The x-axis depicts the belief space $\Delta(S)$, and the y-axis shows the value of each belief. The value function $V_0^*(b)$, i.e., $\max_a \sum_s R(s, a)b(s)$, is indicated by the solid black line and consists of three α_0 vectors, indicated by dashed lines. An example belief $b = (0.75, 0.25)$ is plotted on the x-axis.

next section we will show how to compute optimal piecewise linear and convex (PWLC) value functions that also take into account the future.

In summary, we view the immediate reward as a vector α_0^a for each state and (3.4) averages α_0^a with respect to the belief b. As averaging is a linear operator and V_0^* consists of $|A|$ linear vectors α_0^a, it is piecewise linear. Since the value function is defined as the upper surface of these vectors, due to the max operator, V_0^* is also convex. In the general case, for $h > 0$, we parameterize a value function V_n at stage n by a finite set of vectors or hyperplanes $\{\alpha_n^k\}$, $k = 1, \ldots, |V_n|$. Given a set of vectors $\{\alpha_n^k\}_{k=1}^{|V_n|}$ at stage n, the value of a belief b is given by

$$V_n(b) = \max_{\{\alpha_n^k\}_k} b \cdot \alpha_n^k. \tag{3.5}$$

Additionally, with each vector an action $a(\alpha_n^k) \in A$ is associated, which is the optimal one to take in the current step, for those beliefs for which α_n^k is the maximizing vector. Each vector defines a region in the belief space for which this vector is the maximizing element of V_n. These regions form a partition of the belief space, induced by the piecewise linearity of the value function. The gradient of the value function at b is given by the vector $\alpha_n^b = \arg\max_{\{\alpha_n^k\}_k} b \cdot \alpha_n^k$, and the policy at b is given by $\pi(b) = a(\alpha_n^b)$.

Next, we will show how we can compute V_{n+1} from V_n using H, and that such a backup indeed preserves the piecewise linearity and convexity properties of V_n.

3.3 Bellman backups

In this section we will introduce a basic tool used in POMDP planning: the Bellman backup of a particular belief point. As shown by Smallwood and Sondik (1973), for problems with a finite planning horizon V^* will be piecewise linear and convex, and for infinite-horizon tasks V^* can be approximated arbitrary well by a PWLC value function.

The main idea behind many value-iteration algorithms for POMDPs is that for a given value function V_n and a particular belief point b we can easily compute the vector α_{n+1}^b of HV_n such that

$$\alpha_{n+1}^b = \underset{\{\alpha_{n+1}^k\}_k}{\arg\max}\, b \cdot \alpha_{n+1}^k, \tag{3.6}$$

where $\{\alpha_{n+1}^k\}_{k=1}^{|HV_n|}$ is the (unknown) set of vectors for HV_n. We will denote this operation $\alpha_{n+1}^b = \texttt{backup}(b)$.

To compute $\texttt{backup}(b)$ involves a long series of simple maximizations and sums leading up to (3.18), as we will see next. Starting from (2.13) we have

$$V_{n+1}(b) = \max_a \Big[b \cdot \alpha_0^a + \gamma \sum_o p(o|b,a) V_n(b^{ao}) \Big], \tag{3.7}$$

where b^{ao} is given by (2.11). Substituting (3.5) leads to

$$V_{n+1}(b) = \max_a \Big[b \cdot \alpha_0^a + \gamma \sum_o p(o|b,a) \max_{\{\alpha_n^k\}_k} \sum_{s'} b^{ao}(s') \alpha_n^k(s') \Big]. \tag{3.8}$$

Plugging in the belief update (2.11) for b^{ao} results in

$$\begin{aligned} V_{n+1}(b) = \max_a \Big[& b \cdot \alpha_0^a + \\ & \gamma \sum_o p(o|b,a) \max_{\{\alpha_n^k\}_k} \sum_{s'} \frac{p(o|s',a)}{p(o|b,a)} \sum_s p(s'|s,a) b(s) \alpha_n^k(s') \Big] \end{aligned} \tag{3.9}$$

$$= \max_a \Big[b \cdot \alpha_0^a + \gamma \sum_o \max_{\{\alpha_n^k\}_k} \sum_s b(s) \big[\sum_{s'} p(o|s',a) p(s'|s,a) \alpha_n^k(s') \big] \Big], \tag{3.10}$$

where the $p(o|b,a)$ terms have canceled each other. At this point we define

$$g_{ao}^k(s) = \sum_{s'} p(o|s',a) p(s'|s,a) \alpha_n^k(s'), \tag{3.11}$$

and use the fact that the max operator transfers from the α_n^k to the g_{ao}^k vectors, leading to

$$V_{n+1}(b) = \max_a \Big[b \cdot \alpha_0^a + \gamma \sum_o \max_{\{g_{ao}^k\}_k} b \cdot g_{ao}^k \Big]. \tag{3.12}$$

Using the identity

$$\max_{\{y_j\}_j} x \cdot y_j = x \cdot \arg\max_{\{y_j\}_j} x \cdot y_j \tag{3.13}$$

results in

$$\begin{aligned} V_{n+1}(b) &= \max_a \Big[b \cdot \alpha_0^a + \gamma b \cdot \sum_o \arg\max_{\{g_{ao}^k\}_k} b \cdot g_{ao}^k \Big] &(3.14)\\ &= \max_{\{g_a^b\}_a} b \cdot g_a^b, &(3.15)\\ &\qquad \text{with } g_a^b = \alpha_0^a + \gamma \sum_o \arg\max_{\{g_{ao}^k\}_k} b \cdot g_{ao}^k. &(3.16) \end{aligned}$$

Using (3.13) again leads to

$$V_{n+1}(b) = b \cdot \arg\max_{\{g_a^b\}_a} b \cdot g_a^b. \tag{3.17}$$

Finally, from (3.17) we can derive the vector $\texttt{backup}(b)$, as this is the vector whose inner product with b yields $V_{n+1}(b)$:

$$\texttt{backup}(b) = \arg\max_{\{g_a^b\}_{a \in A}} b \cdot g_a^b, \tag{3.18}$$

with g_a^b defined in (3.16). Note that in general not only the computed α vector is retained, but also which action a was the maximizer in (3.18), as that is the optimal action associated with $\texttt{backup}(b)$.

In the procedure described above we have used the g_{ao}^k vectors (3.11), defined as back-projected copies of α_n^k for each a and o to compute g_a^b (3.16). An equivalent way to compute g_a^b is to take the forward-projected b^{ao}, and to maximize directly over the α_n^k vectors. This alternative derivation starts by applying the identity (3.13) to (3.8):

$$V_{n+1}(b) = \max_a \Big[b \cdot \alpha_0^a + \gamma \sum_o p(o|b,a) b^{ao} \cdot \arg\max_{\{\alpha_n^k\}_k} b^{ao} \cdot \alpha_n^k \Big]. \tag{3.19}$$

We define

$$\alpha_{ao}^b(s') = \arg\max_{\{\alpha_n^k\}_k} b^{ao} \cdot \alpha_n^k, \tag{3.20}$$

and use Bayes' rule (2.11) to rewrite (3.19) as follows:

$$V_{n+1}(b) = \max_a \Big[b \cdot \alpha_0^a + \gamma \sum_o \sum_{s'} p(o|s',a) \sum_s p(s'|s,a) b(s) \alpha_{ao}^b(s') \Big] \tag{3.21}$$

$$V_{n+1}(b) = \max_a \Big[b \cdot \alpha_0^a + \gamma b \cdot \sum_o h_{ao}^b \Big], \text{ where} \tag{3.22}$$

$$h_{ao}^b(s) = \sum_{s'} p(o|s',a) p(s'|s,a) \alpha_{ao}^b(s'). \tag{3.23}$$

Complexity of β(3.24)		Complexity of (3.16)	
(2.11)	$O(\lvert S\rvert^2\lvert A\rvert\lvert O\rvert)$	(3.11)	$O(\lvert S\rvert^2\lvert A\rvert\lvert O\rvert\lvert V\rvert)$
(3.20)	$O(\beta\lvert S\rvert\lvert A\rvert\lvert O\rvert\lvert V\rvert)$	(3.16)	$O(\beta\lvert S\rvert\lvert A\rvert\lvert O\rvert\lvert V\rvert)$
(3.23)	$O(\beta\lvert S\rvert^2\lvert A\rvert\lvert O\rvert)$		
(3.24)	$O(\beta\lvert S\rvert\lvert A\rvert\lvert O\rvert)$		
$O(\lvert S\rvert\lvert A\rvert\lvert O\rvert(\beta\lvert V\rvert+\beta\lvert S\rvert))$		$O(\lvert S\rvert\lvert A\rvert\lvert O\rvert(\beta\lvert V\rvert+\lvert V\rvert\lvert S\rvert))$	

Table 3.1: Complexity of computing an exhaustive backup using either the back-projected g^k_{ao} vectors (3.16) or the forward-projected b^{ao} beliefs (3.24), where β is the number of beliefs that need to be backed up for V_{n+1}.

Analogous to (3.14)-(3.17), we can isolate b, and define

$$g^b_a = \alpha^a_0 + \gamma \sum_o h^b_{ao}. \tag{3.24}$$

As a comparison between (3.16) and (3.24) we will discuss the complexity of computing an exhaustive backup using either. Table 3.1 details the computational complexity of all steps in computing (3.16) and (3.24). When using the forward-projected beliefs, we have to compute (3.24) for enough beliefs necessary for V_{n+1}, resulting in a total time complexity of $O(|S||A||O|(\beta|V|+\beta|S|))$, where β is the number of beliefs that need to be backed up. The complexity of computing a backup using (3.16) is $O(|S||A||O|(\beta|V|+|V||S|))$. The difference is that using the back-projected vectors g^k_{ao}, (3.16) is linear in the size of V_n, while (3.24) is linear in β. Typically β will be larger than $|V_n|$, and as such computing an exhaustive backup using (3.16) is more efficient in general, but the backup defined by (3.24) is more amenable to extensions (see Sec. 4.4.3).

The definition of the `backup` operator involves summing and maximizing over linear functions, which are trivially convex. Since (1) we started with a PWLC V_0, (2) the PWLC property is preserved when summing or maximizing over two PWLC functions, and (3) only a finite number of new vectors can be computed for each V_{n+1} as A and O are finite sets, we can see that by induction the value function for any horizon is also piecewise linear and convex (Smallwood and Sondik, 1973). Without going into detail, for the infinite-horizon case not all V^*_∞ will be PWLC, but those that are not can be approximated arbitrarily close by a PWLC value function, with a bound that depends on γ (Sondik, 1978).

We will now turn to describing a number of value-iteration methods for POMDPs, each of which exploits the PWLC property of the value function.

3.4 Exact value iteration

The Bellman backup operator as introduced in the previous section computes a next-horizon vector for a single belief. Next we will employ this backup operator to compute a complete value function for the next horizon, i.e., one that is optimal for all beliefs in the belief space. Although computing the vector $\texttt{backup}(b)$ for a given b is straightforward, locating the (minimal) set of points b required to compute *all* vectors $\cup_b \texttt{backup}(b)$ of HV_n is very costly. As each b has a region in the belief space in which its α_n^b is maximal, a family of algorithms tries to identify these regions (Sondik, 1971; Cheng, 1988; Kaelbling et al., 1998). The corresponding b of each region is called a "witness" point, as it testifies to the existence of its region. Other exact POMDP value-iteration algorithms do not focus on searching in the belief space. Instead, they consider enumerating all possible vectors of HV_n, followed by pruning useless vectors (Monahan, 1982; Zhang and Liu, 1996; Cassandra et al., 1997; Feng and Zilberstein, 2004; Lin et al., 2004; Varakantham et al., 2005).

3.4.1 Monahan's enumeration algorithm

Let us start by considering the most straightforward way of computing HV_n, due to Monahan (1982). It involves calculating all possible ways HV_n could be constructed, exploiting the known structure of the value function. Note that in each HV_n a finite number of vectors are generated, as we have assumed finite sets A and O. We operate independent of a particular b now so (3.16) can no longer be applied. Instead of maximizing for all $o \in O$ over the g_{ao}^k vectors for the particular b, we now have to include all ways of selecting g_{ao}^k for all o:

$$HV_n = \bigcup_a G_a, \text{ with } G_a = \bigoplus_o G_a^o, \text{ and } G_a^o = \left\{ \frac{1}{|O|}\alpha_0^a + \gamma g_{ao}^k \right\}_k, \qquad (3.25)$$

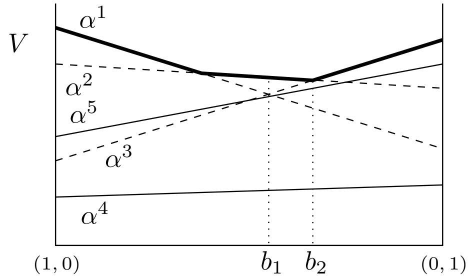
where $\bigoplus$ denotes the cross-sum operator.[1]

Unfortunately, at each stage a finite but exponential number of vectors are generated: $|A||V_n|^{|O|}$. The regions of many of the generated vectors will be empty and these vectors are useless as they will not influence the agent's policy. Technically, they are not part of the value function, and keeping them has no effect on subsequent value functions, apart from the very high computational burden. Therefore, all value-iteration methods in the enumeration family employ some form of pruning. In particular, Monahan (1982) prunes HV_n after computing it:

$$V_{n+1} = \texttt{prune}(HV_n), \qquad (3.26)$$

with HV_n as defined in (3.25). We will now discuss how to implement the $\texttt{prune}$ operator.

[1] Cross sum of sets is defined as: $\bigoplus_k R_k = R_1 \oplus R_2 \oplus \cdots \oplus R_k$, with $P \oplus Q = \{\, p+q \mid p \in P,\ q \in Q \,\}$.



(a) Pruning example.

variables: $\forall s \in S, b(s); x$
maximize: x
subject to:
$b \cdot (\alpha - \alpha') \geq x, \forall \alpha' \in V, \alpha' \neq \alpha$
$b \in \Delta(S)$

(b) Linear program for `prune`.

Figure 3.2: (a) A vector pruning example of a value function V in a two-state POMDP. The y-axis shows the value of each belief, and the x-axis depicts the belief space $\Delta(S)$. Shown are five α vectors, but $\{\alpha^1, \alpha^2, \alpha^3\}$ already suffice for defining V, as the other two have no region for which they are the maximizing α vector. Pruning α^4 is relatively easy and computationally cheap, as it is completely dominated by a single other α vector (Eagle, 1984). However, the pruning of α^5 requires solving a linear program, for instance the one provided in (b). Solving the linear program for $\alpha = \alpha^5$ results in a negative x at b_2, indicating it is dominated and hence can be pruned. Solving for $\{\alpha^1, \alpha^2, \alpha^3\}$ results in positive x values, at $(1,0)$, b_1 and $(0,1)$ respectively, which are witness points as such.

3.4.2 Pruning of α vectors

One way of pruning spurious vectors is to compare each α^k and α^l ($\alpha^k \neq \alpha^l$) in V, and to discard α^k if $\alpha^k(s) \leq \alpha^l(s), \forall s \in S$ (or α^l if $\alpha^l(s) \leq \alpha^k(s), \forall s \in S$) (Eagle, 1984). Such a check at each state is computationally cheap, and will prune α^4 in example value function shown in Fig. 3.2(a), but not α^5. Identifying and subsequently pruning all dominated vectors requires solving a linear program (Bertsimas and Tsitsiklis, 1997). Lark (White, 1991) provided such a linear program, shown in Fig. 3.2(b). It computes for a particular α the belief that has the largest difference x with all other vectors in V. When $x \leq 0$ (or when the linear program is infeasible) there is always another vector or a convex combination of other vectors dominating α, and therefore α can be safely pruned.

Algorithm 3.1 shows an algorithm for pruning a set of vectors G, based on Lark's algorithm (White, 1991). The set G' contains only vectors that have a non-empty witness region, and is constructed incrementally. The algorithm starts with a low-cost check at each corner point e_s of $\Delta(S)$, defined as a vector with all zeros except $e_s(s) = 1$. The maximizing vector of G for each e_s is copied to G', after which we can remove G' from G. For all the vectors remaining in G we solve the linear program described above to move all other non-dominated vectors to G'.

Algorithm 3.1 Pruning a set of vectors G, based on Lark's algorithm.

```
prune(G)
$G' \leftarrow \emptyset$
for all $s \in S$ do
  $\alpha \leftarrow \arg\max_{\alpha' \in G} e_s \cdot \alpha', \quad G' \leftarrow G' \cup \{\alpha\}$
$G \leftarrow G \setminus G'$
while $G \neq \emptyset$ do
  $b, x \leftarrow$ solve linear program Fig. 3.2(b) for $\alpha \in G$ and $V = G'$
  if $x \leq 0$ then
    $G \leftarrow G \setminus \{\alpha\}$
  else
    $\alpha' \leftarrow \arg\max_{\alpha \in G} b \cdot \alpha, \quad G' \leftarrow G' \cup \{\alpha'\}, \quad G \leftarrow G \setminus \{\alpha'\}$
Return $G'$
```

However, the added cost of linear programming is substantial; Cassandra (1998a) reports that on a range of problems solved by exact value iteration, 90% to 95% of the computation time is spent in linear-programming routines, either for pruning or searching the belief space (as in Sec. 3.4.4).

3.4.3 Incremental Pruning

Recall that Monahan (1982)'s algorithm first generates all $|A||V_n|^{|O|}$ vectors of HV_n before pruning all dominated vectors. Incremental Pruning methods (Zhang and Liu, 1996; Cassandra et al., 1997; Feng and Zilberstein, 2004; Lin et al., 2004; Varakantham et al., 2005) save computation time by exploiting the fact that

$$\texttt{prune}(G \oplus G' \oplus G'') = \texttt{prune}(\texttt{prune}(G \oplus G') \oplus G''). \tag{3.27}$$

In this way the number of constraints in the linear program of Fig. 3.2(b) grows slowly (Cassandra et al., 1997), leading to better performance. The basic Incremental Pruning algorithm exploits (3.27) when computing V_{n+1} as follows:

$$V_{n+1} = \texttt{prune}\Big(\bigcup_a G_a\Big), \quad \text{with} \tag{3.28}$$

$$G_a = \texttt{prune}\Big(\bigoplus_o G_a^o\Big) \tag{3.29}$$

$$= \texttt{prune}(G_a^1 \oplus G_a^2 \oplus G_a^3 \oplus \cdots \oplus G_a^{|O|}) \tag{3.30}$$

$$= \texttt{prune}(\cdots \texttt{prune}(\texttt{prune}(G_a^1 \oplus G_a^2) \oplus G_a^3) \cdots \oplus G_a^{|O|}). \tag{3.31}$$

This concludes our brief review of exact value-iteration algorithms that focus on enumerating vectors and pruning dominated ones. In the next section we will shift our focus to algorithms that search the belief space for a set of witness points sufficient for computing an exhaustive backup (Sondik, 1971; Cheng, 1988; Cassandra et al., 1994).

3.4.4 Sondik's One-Pass algorithm

In Sec. 3.3 we have seen that it is straightforward to compute the next-horizon vector α^b_{n+1} for a particular b given the current value function V_n, using the `backup` operator (3.18). Specifically, focusing on a particular belief b reduces the cross-sum of the G_{ao} sets to a simple maximization. We know that V_{n+1} will consist of only a finite number of vectors $\{\alpha^k_{n+1}\}$ (and possibly much less than $|A||V_n|^{|O|}$), and if we knew in advance a witness point b_k for each of them, we could compute $V_{n+1} = \cup_{b_k}\texttt{backup}(b_k)$. This insight gives rise to a series of algorithms that search the belief space for a set $\{b_k\}$, sufficient for generating all vectors of V_{n+1} (Sondik, 1971; Cheng, 1988; Cassandra et al., 1994).

Our focus will be on the One-Pass algorithm (Sondik, 1971; Smallwood and Sondik, 1973), as it is the first and most straightforward algorithm that searches the belief space. It is based on the concept that a particular witness point b_k has a witness region, i.e., the region for which its α^k_{n+1} will dominate all other vectors in V_{n+1}, some of which will still need to be computed. If we could compute b_k's witness region, we could pick a new witness point b'_k just outside b_k's witness region, compute b'_k's witness region and iterate until we have covered the entire belief space. At that point we would have all vectors of V_{n+1}, completing the exhaustive backup stage.

Sondik's One-Pass algorithm is initialized with an empty set $G = \emptyset$ vectors and an empty search list. It starts by backing up an arbitrary belief point b_0. The computed vector α^0_{n+1} is optimal for b_0, but we do not know for which other $b \in \Delta(S)$ it is also optimal. However, without going into details, Sondik defined a number of constraints which define a region (or volume) in the belief space around b_0, and α^0_{n+1} is guaranteed to be optimal for this region. Linear programming is performed to compute the region, and belief points that lie on the edge of the region are added to its search list. The algorithm proceeds by backing up a belief point from its search list, compute its region, add the new edge beliefs to the list, etc., until the regions form a complete partition of the belief space. At that point the algorithm has computed all vectors of V_{n+1}.

However, Sondik's regions are rather conservative, i.e., its borders are still inside the true witness region, which will result in duplicate vectors being computed. Subsequent algorithms that search the belief space such as Relaxed Region, Linear Support (Cheng, 1988) and Witness (Cassandra et al., 1994) improve on the computation time of the One-Pass algorithm by relaxing Sondik's constraints, resulting in fewer but larger regions (while not exceeding the true witness region).

3.4.5 Speeding up exact value iteration with approximate backups

Zhang and Zhang (2001) proposed an alternative approach to exact value iteration, building on the work of Cheng (1988), designed to speed up each exact value-iteration step. It turns out that value iteration still converges to the optimal value function if exact value-update steps are interleaved with approximate update steps in which a new value function V_{n+1} is computed from V_n such that

$$V_n(b) \leq V_{n+1}(b) \leq HV_n(b), \qquad \text{for all } b \in \Delta(S). \tag{3.32}$$

This additionally requires that the value function is appropriately initialized, by choosing V_0 to be a vector with all its components equal to $\frac{1}{1-\gamma} \min_{s,a} R(s,a)$. Such a single vector represents the minimum of cumulative discounted reward obtainable in the POMDP, and is guaranteed to be below V^*. Zhang and Zhang (2001) compute V_{n+1} by backing up witness points of V_n for a number of steps. As we saw above, backing up a set of belief points is a relatively cheap operation. Thus, given V_n, a number of vectors of HV_n are created by applying `backup` to the witness points of V_n, and then a set of linear programs are solved to ensure that $V_{n+1}(b) \geq V_n(b)$, $\forall b \in \Delta(S)$. This is repeated for a number of steps, before an exact value-update step takes place. The authors demonstrate experimentally that a combination of approximate and exact backup steps can speed up exact value iteration.

In general, however, computing optimal planning solutions for POMDPs is an intractable problem for any reasonably sized task, as we discussed in Sec. 2.5 (Papadimitriou and Tsitsiklis, 1987; Madani et al., 2003). This calls for approximate solution techniques. We will describe next a line of research on approximate POMDP algorithms which focus on planning on a fixed set of belief points.

3.5 Approximate value iteration

The major cause of intractability of exact POMDP solution methods is their aim of computing the optimal action for every possible belief point in $\Delta(S)$. For instance, if we use (3.25) we can end up with a series of value functions whose size grows exponentially in the planning horizon. A natural way to sidestep this intractability is to settle for computing an approximate solution by considering only a finite set of belief points. The backup stage reduces to applying (3.18) a fixed number of times, resulting in a small number of vectors (bounded by the size of the belief set). The motivation for using approximate methods is their ability to compute successful policies for much larger problems, which compensates for the loss of optimality.

Approximate POMDP value-iteration methods operating on a fixed set of points are explored by Lovejoy (1991) and in subsequent works (Hauskrecht, 2000; Poon, 2001; Pineau et al., 2003a; Spaan and Vlassis, 2005a). For instance, Pineau et al. (2003a) use an approximate backup operator $\tilde{H}_{\mathrm{PBVI}}$ instead of H, that computes in each value-backup stage the set

$$\tilde{H}_{\mathrm{PBVI}} V_n = \bigcup_{b \in B} \texttt{backup}(b), \tag{3.33}$$

using a fixed set of belief points B. The general assumption underlying these so-called *point-based* methods is that by updating not only the value but also its gradient (the α vector) at each $b \in B$, the resulting policy will generalize well and be effective for beliefs outside the set B. Whether or not this assumption is realistic depends on the POMDP's structure and the contents of B, but the intuition is that in many problems the set of 'reachable' beliefs (reachable by following an arbitrary policy starting from the initial belief) forms a low-dimensional manifold in the belief simplex, and thus can be covered densely enough by a relatively small number of belief points.

Crucial to the control quality of the computed approximate solution is the makeup of B. A number of schemes to build B have been proposed. For instance, one could use a regular grid on the belief simplex, computed, e.g., by Freudenthal triangulation (Lovejoy, 1991). Other options include taking all extreme points of the belief simplex or use a random grid (Hauskrecht, 2000; Poon, 2001). An alternative scheme is to include belief points that can be encountered by simulating the POMDP: we can generate trajectories through the belief space by sampling random actions and observations at each time step (Lovejoy, 1991; Hauskrecht, 2000; Poon, 2001; Pineau et al., 2003a; Spaan and Vlassis, 2005a). This sampling scheme focuses the contents of B to be beliefs that can actually be encountered while experiencing the POMDP model.

The PBVI algorithm (Pineau et al., 2003a) is an instance of such a point-based POMDP algorithm. PBVI starts by selecting a small set of beliefs B_0, performs a number of backup stages (3.33) on B_0, expands B_0 to B_1 by sampling more beliefs, performs again a series of backups, and repeats this process until a satisfactory solution has been found (or the allowed computation time expires). The set B_{t+1} grows by simulating actions for every $b \in B_t$, maintaining only the new belief points that are furthest away from all other points already in B_{t+1}. This scheme is a heuristic to let B_t cover a wide area of the belief space, but comes at a cost as it requires computing distances between all $b \in B_t$. By backing up all $b \in B_t$ the PBVI algorithm generates at each stage approximately $|B_t|$ vectors, which can lead to slow performance in domains requiring large B_t.

In the next section we will present a point-based POMDP value-iteration method which does not require backing up all $b \in B$. We compute backups for a subset of B only, but seeing to it that the computed solution will be effective for the complete set B. As a result we limit the growth of the number of vectors in the successive value-function estimates, leading to significant speedups.

3.6 PERSEUS: a randomized point-based value-iteration method

We have discussed exact and approximate methods for solving POMDPs, which allow us to compute successful plans for agents in stochastic and partially observable environments. Below we describe PERSEUS, an approximate solution method capable of computing competitive solutions in large POMDP domains.

PERSEUS is a point-based value-iteration algorithm for POMDPs (Vlassis and Spaan, 2004; Spaan and Vlassis, 2004, 2005a). The value-update scheme of PERSEUS implements a randomized approximate backup operator $\tilde{H}_{\text{PERSEUS}}$ that increases (or at least does not decrease) the value of all belief points in B. Such an operator can be implemented very efficiently in POMDPs given the shape of the value function. The key idea is that in each value-backup stage we can improve the value of *all* points in the belief set by only updating the value and its gradient of a (randomly selected) subset of the points. In each backup stage, given a value function V_n, we compute a value function V_{n+1} that improves the value of all $b \in B$, i.e., we build a value function $V_{n+1} = \tilde{H}_{\text{PERSEUS}} V_n$ that upper bounds V_n over B (but not necessarily over $\Delta(S)$ which would require linear programming):

$$V_n(b) \leq V_{n+1}(b), \qquad \text{for all } b \in B. \tag{3.34}$$

We first let the agent randomly explore the environment and collect a set B of reachable belief points, which remains fixed throughout the complete algorithm. We initialize the value function V_0 as a single vector with all its components equal to $\frac{1}{1-\gamma} \min_{s,a} R(s,a)$ (Zhang and Zhang, 2001). Starting with V_0, PERSEUS performs a number of backup stages until some convergence criterion is met. The backup stage is defined in Algorithm 3.2.

In PERSEUS, often a small number of vectors will be sufficient to improve $V_n(b)\ \forall b \in B$, especially in the first steps of value iteration. The idea is to compute these vectors in a randomized greedy manner by sampling from increasingly smaller subsets of B. We keep track of the set of non-improved points during a backup stage, consisting of those $b \in B$ whose new value $V_{n+1}(b)$ is still lower than $V_n(b)$. At the start of each backup stage, V_{n+1} is set to $\emptyset$ which means $\tilde{B}$ is initialized to B, indicating that all $b \in B$ still need to be improved in this backup stage. As long as $\tilde{B}$ is not empty, we sample a point b from $\tilde{B}$ and compute $\alpha = \texttt{backup}(b)$. If α improves the value of b (i.e., if $b \cdot \alpha \geq V_n(b)$ at line 3), we add α to V_{n+1} and update $V_{n+1}(b)$ for all $b \in B$ by computing their inner product with the new α. The hope is that α improves the value of many

Algorithm 3.2 Perseus backup stage: $V_{n+1} = \tilde{H}_{\text{Perseus}} V_n$.

```
1: $V_{n+1} \leftarrow \emptyset, \quad \tilde{B} \leftarrow B$
2: Sample belief $b$ uniformly at random from $\tilde{B}$ and compute $\alpha = \texttt{backup}(b)$.
3: if $b \cdot \alpha < V_n(b)$ then
4:     $\alpha = \arg\max_{\{\alpha_n^k\}_k} b \cdot \alpha_n^k$
5: $V_{n+1} \leftarrow V_{n+1} \cup \{\alpha\}$
6: $\tilde{B} = \{b \in B : V_{n+1}(b) < V_n(b)\}$
7: if $\tilde{B} \neq \emptyset$ then
8:     Goto 2
9: Return $V_{n+1}$
```

other points in B, and all these points are removed from $\tilde{B}$. As long as $\tilde{B}$ is not empty we sample belief points from it and add their α vectors.

To ensure termination of each backup stage we have to enforce that $\tilde{B}$ shrinks when adding vectors, i.e., that each α actually improves at least the value of the b that generated it. If not (i.e., $b \cdot \alpha < V_n(b)$ at line 3), we ignore α and insert a copy of the maximizing vector of b from V_n in V_{n+1}. Point b is now considered improved and is removed from $\tilde{B}$ (line 6), together with any other belief points which had the same vector as maximizing one in V_n. This procedure ensures that $\tilde{B}$ shrinks and the backup stage will terminate. A pictorial example of a backup stage is presented in Fig. 3.3.

Perseus performs backup stages until some convergence criterion is met. For point-based methods several convergence criteria can be considered, one could for instance bound the difference between successive value-function estimates $\max_{b \in B}(V_{n+1}(b) - V_n(b))$. Another option would be to track the number of policy changes: the number of $b \in B$ which had a different optimal action in V_n compared to V_{n+1} (Lovejoy, 1991).

3.7 Related Work

In Sec. 3.5 we reported on a class of approximate solution methods for POMDPs that focus on computing a value-function approximation based on a fixed set of prototype belief points. Here we will broaden the picture to other approximate POMDP solution techniques. A related overview is provided by Hauskrecht (2000).

3.7.1 MDP-based heuristics

A few heuristic control strategies have been proposed which rely on a solution $\pi^*(s)$ or $Q^*(s, a)$ of the underlying MDP (Cassandra, Kaelbling, and Kurien, 1996). Perhaps the most straightforward heuristic is for every belief to consider

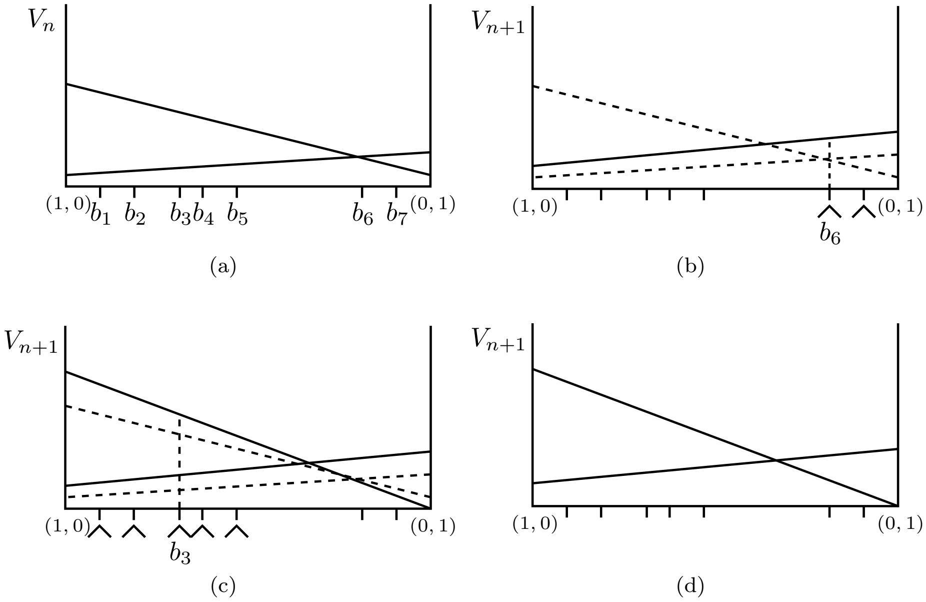


Figure 3.3: Example of a Perseus backup stage in a two-state POMDP. The belief space is depicted on the x-axis and the y-axis represents $V(b)$. Solid lines are α_n^k vectors from the current stage n and dashed lines are α_{n-1}^k vectors from the previous stage. We operate on a B of 7 beliefs, indicated by the tick marks. The backup stage computing V_{n+1} from V_n proceeds as follows: (a) value function at stage n; (b) start computing V_{n+1} by sampling b_6, add $\alpha = \texttt{backup}(b_6)$ to V_{n+1} which improves the value of b_6 and b_7; (c) sample b_3 from $\{b_1, \dots, b_5\}$, add $\texttt{backup}(b_3)$ to V_{n+1} which improves b_1 through b_5; and (d) the value of all $b \in B$ has improved, the backup stage is finished.

its most likely state (MLS), and use the action the MDP policy prescribes for the state

$$\pi_{MLS}(b) = \pi^*(\arg\max_s b(s)). \tag{3.35}$$

The MLS heuristic completely ignores the uncertainty in the current belief, which clearly can be suboptimal.

A simple approximation technique is Q_{MDP} (Littman, Cassandra, and Kaelbling, 1995), which also treats the POMDP as if it were fully observable. Q_{MDP} solves the MDP and defines a control policy

$$\pi_{Q_{\text{MDP}}}(b) = \arg\max_a \sum_s b(s) Q^*(s,a). \tag{3.36}$$

Q_{MDP} can be very effective in some domains, but the policies it computes will not take informative actions, as the Q_{MDP} solution assumes that any uncertainty regarding the state will disappear after taking one action. As such, Q_{MDP} policies will fail in domains where repeated information gathering is necessary. Cassandra (1998a) provides an extensive experimental comparison of MLS, Q_{MDP}, and other MDP-based heuristics.

One can also expand the MDP setting to model some form of uncertainty without considering full-blown POMDP beliefs. For instance, in robotics the navigation under localization uncertainty problem can be modeled by the mean and entropy of the belief distribution (Cassandra et al., 1996; Roy and Thrun, 2000). Although attractive from a computational perspective, such approaches are likely to fail when the belief is not uni-modal but has a more complex shape.

3.7.2 Grid-based approximations

One way to sidestep the intractability of exact POMDP value iteration is to grid the belief simplex, using either a fixed grid (Drake, 1962; Lovejoy, 1991; Bonet, 2002) or a variable grid (Brafman, 1997; Zhou and Hansen, 2001). Value backups are performed for every grid point, but only the value of each grid point is preserved and the gradient is ignored. The value of non-grid points is defined by an interpolation rule. The grid based methods differ mainly on how the grid points are selected and what shape the interpolation function takes. In general, regular grids do not scale well in problems with high dimensionality and non-regular grids suffer from expensive interpolation routines.

3.7.3 Policy search

An alternative to computing an (approximate) value function is policy search: these methods search for a good policy within a restricted class of controllers (Platzman, 1981). For instance, policy iteration (Hansen, 1998b) and bounded policy iteration (BPI) (Poupart and Boutilier, 2004) search through the space of (bounded-size) stochastic finite-state controllers by performing policy-iteration steps. Other options for searching the policy space include gradient ascent (Meuleau, Kim, Kaelbling, and Cassandra, 1999a; Kearns, Mansour, and Ng, 2000; Ng and Jordan, 2000; Baxter and Bartlett, 2001; Aberdeen and Baxter, 2002) and heuristic methods like stochastic local search (Braziunas and Boutilier, 2004). In particular, the Pegasus method (Ng and Jordan, 2000) estimates the value of a policy by simulating a (bounded) number of trajectories from the POMDP using a fixed random seed, and then takes steps in the policy space in order to maximize this value. Policy search methods have demonstrated success in several cases, but searching in the policy space can often be difficult and prone to local optima.

3.7.4 Heuristic search

Another approach for solving POMDPs is based on heuristic search (Satia and Lave, 1973; Hansen, 1998a; Smith and Simmons, 2004). Defining an initial belief b_0 as the root node, these methods build a tree that branches over (a, o) pairs, each of which recursively induces a new belief node. These methods bear a similarity to PERSEUS since they also focus on reachable beliefs from b_0. However, they differ in the way belief points are selected to back up; in the above methods branch-and-bound techniques are used to maintain upper and lower bounds to the expected return at fringe nodes in the search tree. Hansen (1998a) proposes a policy-iteration method that represents a policy as a finite-state controller, and which uses the belief tree to focus the search on areas of the belief space where the controller can most likely be improved. However, its applicability to large problems is limited by its use of full dynamic-programming updates. HSVI (Smith and Simmons, 2004) is an approximate value-iteration technique that performs a heuristic search through the belief space for beliefs at which to update the bounds, similar to work by Satia and Lave (1973). An alternative approach to maintaining uncertainty estimates of an approximate value function is based on Gaussian Processes (Tuttle and Ghahramani, 2004).

3.7.5 Compression techniques

Compression techniques can be applied to large POMDPs to reduce the dimensionality of the belief space, facilitating the computation of an approximate solution. Roy, Gordon, and Thrun (2005) apply Exponential family PCA to a sample set of beliefs to find a low-dimensional representation, based on which an approximate solution is sought. Such a nonlinear compression can be very effective, but requires learning a reward and transition model in the reduced space. After such a model is learned, one can compute an approximate solution for the original POMDP using, e.g., MDP value iteration. Alternatively linear compression techniques can be used which preserve the shape of value function (Poupart and Boutilier, 2003a). Such a property is desirable as it allows one to exploit the existing POMDP machinery. For instance, linear compression has been applied as a preprocessing step for BPI (Poupart and Boutilier, 2005) as well as PERSEUS (Poupart, 2005).

3.8 Experimental results

We will show experimental results applying PERSEUS on benchmark problems from the POMDP literature, and a robot planning domain, TRC. Table 3.2 summarizes these domains in terms of the size of S, O and A. Each belief set was gathered by simulating trajectories of interactions of the agent with the POMDP environment starting at a random state sampled from b_0, and at each

Name	$\lvert S \rvert$	$\lvert O \rvert$	$\lvert A \rvert$
Tiger-grid	33	17	5
Hallway	57	21	5
Hallway2	89	17	5
Tag	870	30	5
TRC	1000	10	4

Table 3.2: Characteristics of problem domains.

time step the agent picked an action uniformly at random. In all domains the discount factor γ was set to 0.95.

3.8.1 Benchmark domains

The Hallway, Hallway2 and Tiger-grid problems (introduced by Littman et al., 1995) are maze domains that have been commonly used to test scalable POMDP solution techniques (Littman et al., 1995; Brafman, 1997; Zhou and Hansen, 2001; Pineau et al., 2003a; Smith and Simmons, 2004; Spaan and Vlassis, 2005a; Poupart, 2005). The Tag domain (Pineau et al., 2003a) is an order of magnitude larger than the first three problems, and is a recent benchmark problem (Pineau et al., 2003a; Smith and Simmons, 2004; Braziunas and Boutilier, 2004; Poupart and Boutilier, 2004; Spaan and Vlassis, 2005a; Poupart, 2005).

3.8.1.1 Benchmark Mazes

Littman et al. (1995) introduced three benchmark maze domains: Tiger-grid, Hallway, and Hallway2. All of them are navigation tasks: the objective for an agent is to reach a designated goal state as quickly as possible. The agent observes each possible combination of the presence of a wall in four directions plus a unique observation indicating the goal state; in the Hallway problem three other landmarks are also available. At each step the agent can take one out of five actions: {*stay in place, move forward, turn right, turn left, turn around*}. Both the transition and the observation model are noisy. Table 3.3(a) through (c) compares the performance of Perseus to other algorithms. For each problem we sampled a set B of 1,000 beliefs, and executed Perseus 10 times for each problem using different random seeds. The average expected discounted reward R is computed from 1,000 trajectories starting from random states (drawn according to b_0) for each of the 10 Perseus runs, and following the computed policy. The reported reward R is the average over these 10,000 trajectories. Perseus reaches competitive control quality using a small number of vectors resulting in a considerable speedup.[1]

[1] Perseus and Q_{MDP} results were computed in Matlab on an Intel Pentium IV 2.4 GHz; other results were obtained on different platforms, so time comparisons are rough.

Tiger-grid	R	$\lvert\pi\rvert$	T
HSVI	2.35	4860	10341
Perseus	2.34	134	104
PBUA	2.30	660	12116
PBVI	2.25	470	3448
BPI w/b	2.22	120	1000
Grid	0.94	174	n.a.
Q_{MDP}	0.23	n.a.	2.76

(a) Results for Tiger-grid.

Hallway	R	$\lvert\pi\rvert$	T
PBVI	0.53	86	288
PBUA	0.53	300	450
HSVI	0.52	1341	10836
Perseus	0.51	55	35
BPI w/b	0.51	43	185
Q_{MDP}	0.27	n.a.	1.34

(b) Results for Hallway.

Hallway2	R	$\lvert\pi\rvert$	T
Perseus	0.35	56	10
HSVI	0.35	1571	10010
PBUA	0.35	1840	27898
PBVI	0.34	95	360
BPI w/b	0.32	60	790
Q_{MDP}	0.09	n.a.	2.23

(c) Results for Hallway2.

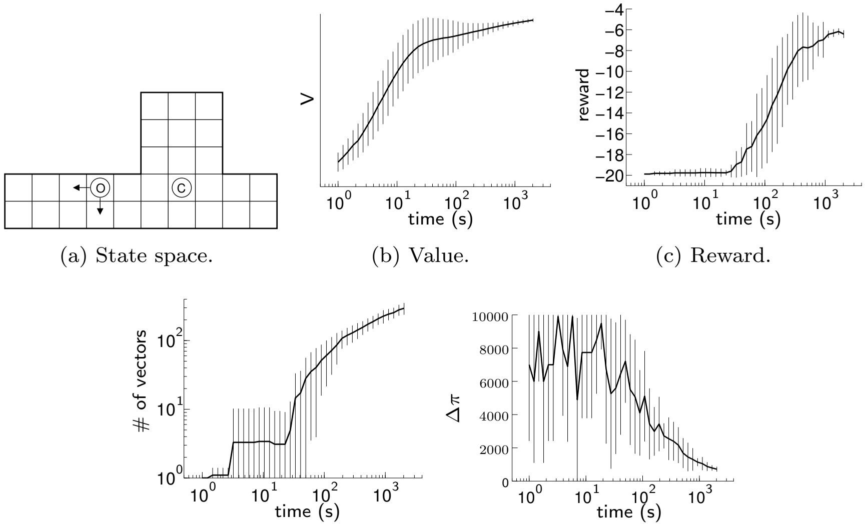
Tag	R	$\lvert\pi\rvert$	T
Perseus	−6.17	280	1670
HSVI	−6.37	1657	10113
BPI w/b	−6.65	17	250
BBSLS	≈ -8.3	30	10^5
BPI n/b	−9.18	940	59772
PBVI	−9.18	1334	180880
Q_{MDP}	−16.9	n.a.	16.1

(d) Results for Tag.

Table 3.3: Experimental comparisons of Perseus with other algorithms. Perseus results are averaged over 10 runs. Each table lists the method, the average expected discounted reward R, the size of the solution $|\pi|$ (value function or controller size), and the time T (in seconds) used to compute the solution. Sources: PBVI (Pineau et al., 2003a), BPI no bias (Poupart and Boutilier, 2004), BPI with bias (Poupart, 2005), HSVI (Smith and Simmons, 2004), Grid (Brafman, 1997), PBUA (Poon, 2001) and BBSLS (Braziunas and Boutilier, 2004) (approximate, read from figure).

3.8.1.2 Tag

The goal in the Tag domain, described by Pineau et al. (2003a), is for a robot to search for a moving opponent robot and tag it. The chasing robot cannot observe the opponent until they occupy the same position, at which time it should execute the *tag* action in order to win the game, and receive a reward of 10. If the opponent is not present at the same location, the reward will be −10, and the robot is penalized with a −1 reward for each motion action it takes. The opponent tries to escape from being tagged by moving away of the chasing robot, however, it has a 0.2 probability of remaining at its location. The chasing and opponent robot both start at a random location. The chasing robot has perfect information regarding its own position and its movement actions {*north, east, south, west*} are deterministic. The state space is represented as



(a) State space. (b) Value. (c) Reward.

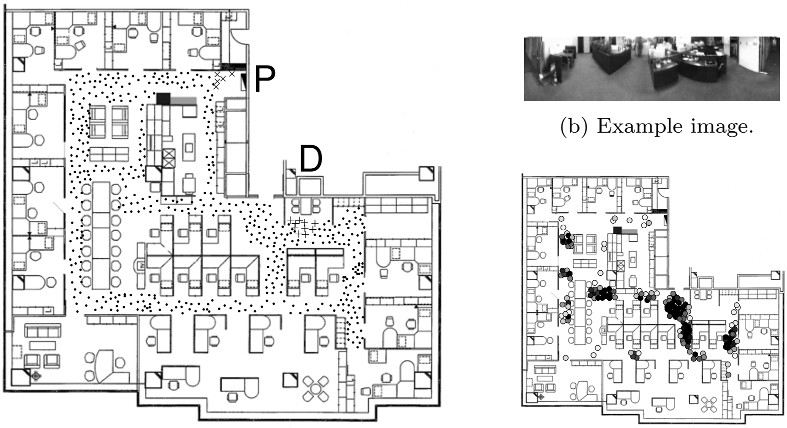
(d) No. of vectors. (e) Policy changes.

Figure 3.4: Tag domain. (a) State space with chasing and opponent robot; (b)–(e) performance of Perseus.

the cross-product of the states of the two robots. Both robots can be located in one of the 29 positions depicted in Fig. 3.4(a), and the opponent can also be in a *tagged* state, resulting in a total of 870 states. Tag is a rather large benchmark problem compared to other POMDP problems studied in literature, but it exhibits a sparse structure. We applied Perseus to a belief set B of 10,000 points.

In Fig. 3.4(b)–(e) we show the performance of Perseus averaged over 10 runs, where error bars indicate standard deviation within these runs. To evaluate the computed policies we tested each of them on 10 trajectories (of at most 100 steps) times 100 starting positions (sampled from the starting belief b_0). Fig. 3.4(b) displays the value as estimated on B, $\sum_{b \in B} V(b)$; (c) the expected discounted reward averaged over the 1,000 trajectories; (d) the number of vectors in the value-function estimate, $|\{\alpha_n^k\}|$; and (e) the number of policy changes: the number of $b \in B$ which had a different optimal action in V_{n-1} compared to V_n. The latter can be regarded as a measure of convergence for point-based solution methods (Lovejoy, 1991). We can see that in almost all experiments Perseus reaches solutions of virtually equal quality and size.

Table 3.3(d) compares the performance of Perseus with other state-of-the-art methods. The results show that in the Tag problem Perseus displays better control quality than any other method and computes its solution an order of



(b) Example image.

(a) Environment.

(c) Observation model.

Figure 3.5: TRC domain. (a) The TRC environment, in which the markers $\{\cdot, +, \times\}$ denote the grid positions of the problem. The positions $\times$ (close to P) are the pickup locations, the ones marked by $+$ (below D) are the delivery (goal) positions. (b) An example panoramic images, corresponding to a particular observation o_k. (c) The induced $p(x|o_k)$, where the darker the disk, the higher the probability.

magnitude faster than most other methods. Specifically, its solution computed on $|B| = 10{,}000$ beliefs consists of only 280 vectors, much less than PBVI which maintains a vector for each of its 1334 $b \in B$. This indicates that the randomized backup stage of Perseus is justified: it takes advantage of a large B while the size of the value function grows moderately with the planning horizon, leading to significant speedups. It is interesting to compare the two variations of BPI, with bias (w/b) (Poupart, 2005) or without (n/b) (Poupart and Boutilier, 2004). The bias focuses on the reachable belief space by incorporating the initial belief which dramatically increases its performance in solution size and computation time, but it does not reach the control quality of Perseus.

3.8.2 TRC robot planning domain

We applied Perseus on a problem based on data obtained in a realistic setting. The TRC domain is a delivery task in an office environment with 1000 states (Spaan and Vlassis, 2004). The task is to pick up mail at the entrance of the office (P, see Fig. 3.5(a)) and deliver it to a certain room (D). At the start of the task the robot is uncertain about its location, and whether it has picked up the mail. The observation model is based on panoramic images taken by an omnidirectional camera mounted on the robot. We used the MEMORABLE[1] robot database that contains a set of approximately 8000 images collected manually by driving the robot around in a 17 × 17 meters office environment with constant orientation, with a sampling resolution of 0.1 meters. Fig. 3.5(b) shows an example image from this database.

As we assume finite and discrete sets S and O in this chapter we need to discretize the state space and the observation space. The state space is defined as the cross-product of the robot's location x and a single bit, indicating whether the robot has already successfully picked up the mail which needs delivery. For discretizing the positions in the map of the environment we performed a k-means clustering (Likas, Vlassis, and Verbeek, 2003) on a subset of all possible positions, resulting in a grid of 500 positions $X = \{x_k\}$, which are depicted in Fig. 3.5(a).

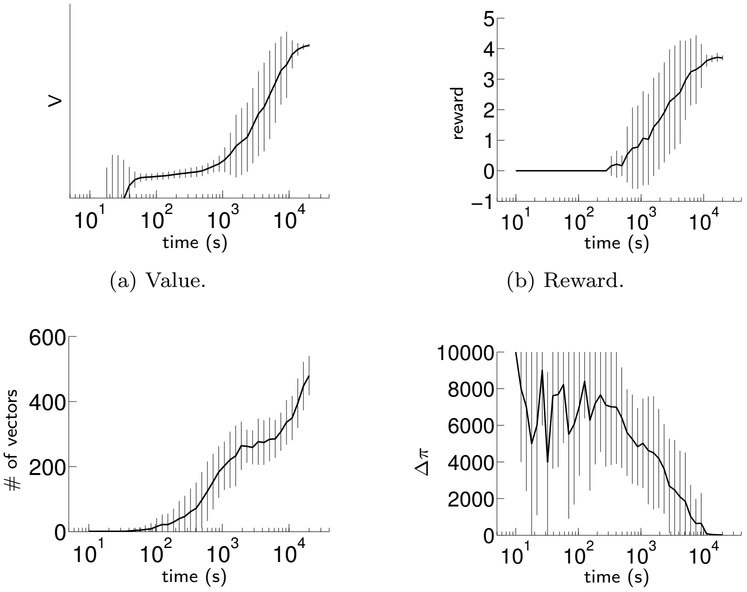
For the discretization of the observation space one should choose the number of prototype observations carefully. A large number of observations can provide more discriminant information on the true position of the robot and thus lead to more peaked beliefs. The more peaked the reachable beliefs are, the more the problem resembles the underlying MDP and the easier it would be to find a good policy. On the other hand, a large number of observations will not solve the problem of perceptual aliasing. Furthermore, the number of observation prototypes determines the size of the set $\{\alpha_{n+1}^k\}$ used in the `backup` operations in (3.18). As such, the number of observation prototypes increases the computational requirements of the algorithm. We applied Principal Component Analysis (PCA) on the image data in order to reduce their dimensionality (Vlassis, Terwijn, and Kröse, 2002). We computed a three-dimensional feature vector for each one of a (randomly chosen) set of 1000 images, by projecting them to the first three eigenvectors (those with the largest eigenvalues) of their covariance matrix. Finally, to discretize this feature space, we used k-means clustering resulting in 10 three-dimensional prototype feature vectors $\{o_1, \ldots, o_{10}\}$.

We constructed the discrete observation model $p(o|s)$ as follows (where we dropped the dependence on a for simplicity). We projected each image in the database to the feature space and found its nearest prototype feature vector among $\{o_k\}_{k=1\ldots10}$. We also associated each robot position in the database with its nearest prototype position in X. Then the probability of observing a prototype feature vector o_i from a prototype robot location x_k can be computed by a histogram operation as follows

$$p(o_k|x_k) = \frac{p(o_k, x_k)}{\sum_{o_l} p(o_l, x_k)} \tag{3.37}$$

where $p(o_k, x_k)$ is simply the fraction of pairs $\{o_i, x_i\}$ in the database such that $o_i \in o_k$ and $x_i \in x_k$, where '$\in$' denotes nearest-cluster membership. Finally,

[1] The MEMORABLE database has been provided by the Tsukuba Research Center in Japan, for the Real World Computing project.



(a) Value.

(b) Reward.

(c) No. of vectors.

(d) Policy changes.

Figure 3.6: Results for the TRC domain.

we duplicated this model for both instances of the bit indicating whether the mail has been picked up. Fig. 3.5(b) displays a panoramic images closest to a particular $o_k \in O$, Fig. 3.5(c) shows the corresponding $p(x|o_k)$.

The robot can execute four basic motion commands {*north, east, south, west*} which transports it according to a Gaussian distribution centered on the expected resulting position (translated two meters in the corresponding direction), with the probability mass distributed over the discrete states. In order to accomplish its task the robot must first execute the *pickup* action in one of the five pickup states near the entrance of the office (marked by P in Fig. 3.5(a)). Only in one of these states the action results in flipping the pickup bit, after which delivering the mail has become possible. To complete the task and receive a reward of 10 the robot has to execute the *delivery* in one of the ten delivery states (indicated by D in Fig. 3.5(a)). Trying to deliver without the mail or delivering to the wrong location is penalized with a reward of -10. Attempting to pick up the mail outside the pickup locations is penalized with reward -1. All motion actions yield no reward.

We sampled a belief set B of $10,000$ belief points and we ran Perseus 10 times with different random seeds. To evaluate the computed value-function estimates we collected rewards by sampling trajectories from 100 random starting

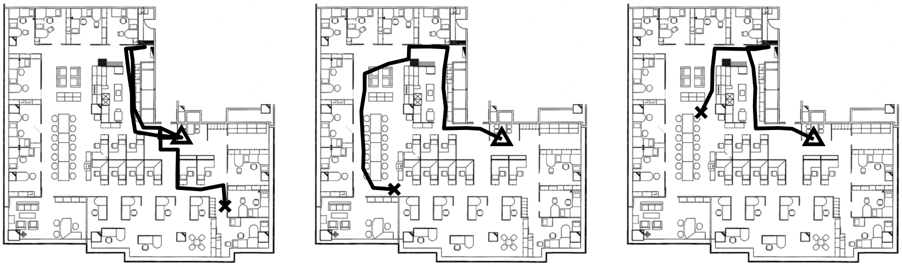


Figure 3.7: Some example trajectories in the TRC environment. Start positions are marked with × and the last state of each trajectory is denoted by a △.

locations with the pickup bit off. Note that the robot has no knowledge regarding the pickup bit until it has picked up the mail at the appropriate location. In our experiments we used a discount factor $\gamma = 0.95$ and each trajectory was stopped after 100 steps (if the robot had not yet delivered the mail by then).

Fig. 3.6 shows the good performance of our algorithm, the error bars indicate standard deviation within the 10 runs of the algorithm. Fig. 3.6(a) displays the value as estimated on B, $\sum_{b \in B} V(b)$, (b) the expected discounted reward, (c) the number of vectors in the value-function estimate, $|\{\alpha_n^k\}|$ and (d) the number of policy changes: the number of $b \in B$ that have a different optimal action in V_n compared to V_{n-1}. We can see that algorithm converges to approximately the same solution quality for all runs, both in value and collected reward. The amount of policy changes drops to below 0.5% of B, which can also be regarded as an indication of convergence. The number of vectors in the value-function estimates grows as the planning horizon increases but the size of the value function remains acceptable. To visualize the policies that our algorithm computes, we plotted some example trajectories in Fig. 3.7. They show the computed policy directs the robot to first move to the pickup states, pick up the mail, and then move to the delivery locations in order to deliver the mail. We also tested Q_{MDP} on this problem, but it fails to compute a successful policy (it receives reward 0) due to the fact that it cannot represent the uncertainty regarding the pickup bit.

3.9 Discussion

The key observation underlying the Perseus algorithm is that when a belief b is backed up, the resulting vector improves not only $V(b)$ but often also the value of many other belief points in B. This results in value functions with a relatively small number of vectors (as compared to, e.g., Poon, 2001; Pineau et al., 2003a). Experiments show indeed that the number of vectors grows modestly with the number of backup stages ($|V_n| \ll |B|$). In practice this means that we can afford to use a much larger B than other point-based methods, which has a positive effect on the approximation accuracy as dictated by the bounds of Pineau et al. (2003a). Furthermore, compared with other methods that build the set B based on various heuristics (Pineau et al., 2003a; Smith and Simmons, 2004), our build-up of B is cheap as it only requires sampling random trajectories starting from b_0. Moreover, duplicate entries in B will only affect the probability that a particular b will be sampled in the value-update stages, but not the size of V_n.

An alternative to using a single fixed set B that is collected by following a fixed policy at the beginning of the algorithm, would be to resample a new B_t after every t-th backup stage (or at fixed intervals) by following the most recent policy. Such an approach could be justified by the fact that an agent executing an optimal policy will most probably visit only a (small) subset of the beliefs in B. We have not tested how such a scheme would affect the solution quality of Perseus and what trade-offs we can achieve for the additional computational cost of sampling multiple sets B. We note that similar 'off-policy' learning using a fixed set of sampled states has also been adopted by other recent algorithms like LSPI (Lagoudakis and Parr, 2003) and PSDP (Bagnell, Kakade, Ng, and Schneider, 2004).

The backups of Perseus on a fixed set B can be viewed as a particular instance of asynchronous dynamic programming (Bertsekas and Tsitsiklis, 1989). In asynchronous dynamic-programming algorithms no full sweeps over the state space are made, but the order in which states are backed up is arbitrary. This allows an algorithm to focus on backups which may have a high potential impact, as for instance in the prioritized-sweeping algorithm for solving fully observable MDPs (Moore and Atkeson, 1993; Peng and Williams, 1993). A drawback is that the notion of an exact planning horizon is somewhat lost: in general, after performing n backup stages the computed plan will not be considering n steps into the future, but less. By backing up non-improved belief points asynchronously Perseus focuses on interesting regions of the (reachable) belief space, and by sampling at random ensures that eventually all $b \in B$ will be taken into account. As we ensure that the value of a particular belief point never decreases, we are guaranteed that Perseus will converge: the proof only requires observing that every added vector is always below V^* (Poon, 2001; Vlassis and Spaan, 2004). Moreover, as we explained above, Perseus can handle large belief sets B, thus obviating the use of dynamic belief-point selection strategies like those proposed by Hauskrecht (2000); Poon (2001), and Pineau et al. (2003a). Note that the only parameter to be set by the user is the size of B; however, the complexity of the resulting policy seems to be only mildly dependent on the size of B.

An interesting issue is how many new vectors are generated in each backup stage of Perseus, and how this may affect the speed of convergence of the algorithm. In general, the smaller the size $|V_n|$ of a value function, the faster the backups (since the `backup` operator has linear dependence on $|V_n|$). On the other hand, two consecutive value functions may differ arbitrarily in size—and we have observed cases where the new value function has fewer vectors than the old value function—which makes it hard to derive bounds on the speed of convergence of PERSEUS and complicates the analysis of the involved trade-offs. We have mainly identified two cases where only a small number of new vectors are added to a value function. The first case is during the initial backup stages, and when V_0 has been initialized very low (e.g., for large γ and large negative immediate reward). In this case a single vector may improve all points, for a number of backup stages, until the value function has reached some sufficient level. The second case is near convergence, when the value function has almost converged in certain regions of the belief space. Sampling a belief point in such a region will result in a (near) copy of the old vector. Whereas the former case provides evidence that the value function has been initialized too low (and adding a single vector is an efficient way to 'correct' this), the latter case may be viewed as providing evidence for the convergence of PERSEUS.

Chapter 4

Planning under uncertainty in continuous domains

In the previous chapter we introduced PERSEUS, our POMDP algorithm for planning in stochastic and partially observable environments. We assumed that the agent's universe was discrete and finite, as in most of the POMDP literature. This chapter will broaden the applicability of PERSEUS by extending it to continuous state and action spaces.

4.1 Introduction

The majority of literature on model-based solution techniques for POMDPs considers discrete states, actions and observations. Unfortunately, many real-world POMDPs are naturally defined with continuous states, actions and observations. Returning to the service-robot domain we considered in Chapter 2, we can see that many of the elements of the problem are naturally modeled as continuous quantities. For instance, the robot's location can be denoted by a pair of real numbers indicating its (x, y) position on some Cartesian coordinate system, and the movement actions it executes can be modeled as driving k meters at heading θ, where k and θ are again real numbers. Instead of discretizing the state and action sets, in this chapter we will work directly in the continuous spaces.

First, in Sec. 4.2 we extend PERSEUS to compute plans for agents which have a continuous (or very large discrete) set of actions at their disposal (Spaan and Vlassis, 2005b,a). Few POMDP solution techniques work directly with continuous action spaces, but Thrun (2000) applied a particle filter to a POMDP with continuous state and action space, and certain policy search methods can tackle continuous action spaces directly (Ng and Jordan, 2000; Baxter and Bartlett, 2001). Instead of discretizing the action space we sample actions from it, in a way that is consistent with the Perseus backup stage. When we sample a belief b from the belief set to backup, we also sample a number of actions from the action space. We treat the sampled action set as any discrete action space, and backup b using the sampled action set. If none of the sampled actions is appropriate for the particular belief, i.e., the value of b is not improved, we copy the old vector associated with b, as in the regular Perseus backup stage (see Algorithm 3.2). In this way we ensure convergence, and in our experiments we explore different ways of sampling a set of actions. Without any prior knowledge, we can sample actions uniformly at random, but we can also exploit the knowledge of the best known action a sofar (the action parameters associated with a belief's vector in the current value function), for example by sampling actions in a's neighborhood. In particular, we can sample exploring and exploiting actions in at the same time, and keep the best one.

Next, we propose a method for dealing with continuous-state POMDPs (Porta, Spaan, and Vlassis, 2005). The main difficulty in designing algorithms for continuous-state POMDPs is that integrals, the generalization of the sums for discrete state spaces, can rarely be computed in closed form. We need to be able to compute these integrals at many points throughout the value-iteration procedure, for instance for computing a belief update, or to compute the expected value of a particular belief given a value function. We demonstrate in Sec. 4.3.1 that value functions defined over the infinite-dimensional belief states induced by continuous states are piecewise linear and convex (PWLC). As we saw in Chapter 3, we can take advantage of the PWLC shape of the value function when designing exact and approximate solution methods. In Sec. 4.3.2 we extend the Perseus algorithm to work with linear combinations of Gaussian distributions as a representation for value functions and belief states.

In this chapter, we also describe work by Hoey and Poupart (2005) that extends Perseus to continuous observation spaces. Combining the three extensions will result in a POMDP planner for fully continuous domains. In Sec. 4.5 we report on experiments for both extensions presented in this chapter. First we consider two discrete-state domains, one in which an agent equipped with proximity sensors can move at a continuous heading and distance, and we present results from a navigation task involving a mobile robot with omnidirectional vision in a perceptually aliased office environment. Next we consider a continuous-state domain, in which we demonstrate our Perseus extension to continuous state spaces on a similar robot navigation task.

4.2 Continuous action spaces

An attractive feature of Perseus is that it can be naturally extended to very large or continuous action spaces, due to the 'improve–only' principle of its backup stage. Note that the `backup` operator, repeated here for convenience, involves a maximization over all actions $a \in A$:

$$\texttt{backup}(b) = \arg\max_{\{g_a^b\}_{a \in A}} b \cdot g_a^b, \tag{3.18}$$

where the vectors g_a^b are defined in (3.16) or (3.24). For very large or continuous action spaces, the full maximization over actions in (3.18) is clearly infeasible, but one can resort to sampling-based techniques. The idea here is to replace the full maximization over actions with a *sampled max* operator that performs the maximization over a random subset of A (Szepesvári and Littman, 1996). The use of such a sampled max operator is very well suited for the backup scheme of PERSEUS in which we only require that the values of belief points do not decrease over two consecutive backup stages. In particular, we can replace the `backup` operator in (3.18) with a new backup operator $\alpha = \texttt{backup}'(b)$ defined as follows (Spaan and Vlassis, 2005b,a):

$$\texttt{backup}'(b) = \arg\max_{\{g_a^b\}_{a \in A'_b}} b \cdot g_a^b, \tag{4.1}$$

where A'_b is a random set of actions drawn from A, and g_a^b as above.

The `backup'` operator can simply replace the `backup` operator in line 2 of Algorithm 3.2 (see p. 42). As in the full maximization case, we need to check in line 3 whether any of the vectors generated by the actions in A'_b improves the value of the particular belief point. If not, we keep the old vector with its associated action that was selected in a previous backup stage. Concerning the sample complexity of the `backup'` operator, i.e., how often we need to sample to improve $V(b)$, we can derive simple bounds that involve the number of actions drawn and the probability to find a 'good' action from A (good in terms of value improvement of b). We can easily show that with probability at least $1-\delta$, the best action among $k = |A'_b|$ actions selected uniformly at random from A is among the best ϵ fraction of all actions from A, if $k \geq \lceil \log\delta / \log(1-\epsilon) \rceil$. The proof involves the observation that the probability that none of the k samples is in the best ϵ fraction is $(1-\epsilon)^k$.

In practice various sampling schemes are possible, which vary in the way A'_b is constructed. We have identified a number of proposal distributions from which to sample actions: (1) uniform from A, (2) a Gaussian distribution centered on the best known action for the particular b, i.e., $a(\alpha_n^b)$, and (3) a Dirac distribution on $a(\alpha_n^b)$. The latter two take into account the policy computed so far by focusing on the current action associated with the input belief b (as recorded in V_n), while sampling uniformly at random uses no such knowledge. Actions sampled uniformly at random can be viewed as exploring actions, while the other two distributions are exploiting current knowledge. As we can select the makeup of A'_b, we can choose any combination of the distributions mentioned above, allowing us to consider exploring and exploiting actions at the same time. In our experiments (see Sec. 4.5.1) we implement the `backup′` operator using a number of different combinations and analyze their effects.

An alternative to sampling actions is to discretize the action space. A computational advantage of reducing the action space to a fixed set of discrete actions is the fact that when A is small enough one can cache in advance the explicit tabular representation of the transition, observation and reward models for all $a \in A$. In contrast, when we sample a real-valued action we have to generate from a parametric description of these models their tabular representation, necessary for instance for computing the back-projected vectors (3.11) used in the backup operator. However, discretization has its limitations, particularly when considering scalability. Using a naive uniform discretization, the number of discrete actions grows exponentially with the number of dimensions of the action space. For instance, consider a robotic arm with a large number of joints or a robot which can control a number of sensors simultaneously: discretization would require a number of bins that is exponential in the number of joints or sensors, respectively. More involved discretization strategies such as quadtrees (Samet, 1984) or non-uniform resolution grids (Ferguson and Stentz, 2006) can save memory, but come at additional overhead. In the experiments of Sec. 4.5.1.2 we will compare discretizing the action space to our sample-based approach.

4.3 Continuous-state POMDPs

In Chapter 3 we described several algorithms for exact or approximate solving of POMDPs with discrete state spaces that rely on a piecewise linear and convex (PWLC) value function, a representation based on a discrete set of supporting vectors. In this section, we show that this representation can be generalized to continuous-state POMDPs, while assuming finite and discrete sets of actions and observations (Porta et al., 2005). However, our extension to continuous actions, as detailed in Sec. 4.2, is rather orthogonal and could be applied to continuous-state POMDPs in a straightforward manner. Point-based POMDP solvers such as Perseus have been extended to continuous observation spaces by Hoey and Poupart (2005), but here we will focus on discrete observation spaces for simplicity reasons.

We prove that the value function for continuous-state POMDPs is PWLC on a set of α functions that play the same role of the α vectors in discrete-state POMDPs. Moreover, we can also prove that the value-function recursion is isotonic and a contraction (Porta, Spaan, and Vlassis, 2004). From these results follows that the continuous-state value-function recursion is convergent to a single fixed point and, therefore, from such an approximation to the optimal value function V^*, we can derive (near) optimal policies (see Puterman, 1994, Theorems 6.3.1 and 6.2.3). Using the theoretical results presented in Sec. 4.3.1, we will define an approximate value-iteration algorithm for continuous-state POMDPs in Sec. 4.3.2.

4.3.1 Value functions for continuous-state POMDPs

First we will generalize a number of concepts introduced in Chapter 2. The propagation of a belief b through the transition model is defined for the continuous case as

$$p(s'|b,a) = \int_{s\in S} p(s'|s,a)b(s). \tag{4.2}$$

The Bellman equation (2.13) generalizes to continuous state spaces as follows:

$$V^*(b) = \max_a \Big[\langle R_a, b\rangle + \gamma \sum_o p(o|b,a)V^*(b^{ao})\Big], \tag{4.3}$$

where the $\langle f, b\rangle$ operator is defined as the expectation of a given function $f : S \to \mathbb{R}$ with respect to the belief b, and R_a is the reward function for executing action a. For POMDPs with discrete state spaces, this expectation can be written as the inner product

$$\langle f, b\rangle = \sum_{s\in S} f(s)b(s), \tag{4.4}$$

as we have seen in Chapter 2. For continuous-state POMDPs, the expectation is an integral over S:

$$\langle f, b\rangle = \int_{s\in S} f(s)b(s). \tag{4.5}$$

In both cases, the operator $\langle f, b\rangle$ is a linear function in b.

We now prove that the continuous-state POMDP value function is PWLC over the belief space.

Theorem 4.1. The value function in a continuous-state POMDP can be expressed as

$$V_n(b) = \max_{\{\alpha_n^k\}_k} \langle \alpha_n^k, b\rangle, \tag{4.6}$$

for appropriate α functions $\alpha_n^k : S \to \mathbb{R}$.

Proof. The proof, as in the discrete case (see Sec. 3.3), is done via induction. For planning horizon 0, we only have to take into account the immediate reward and, thus, we have that

$$V_0(b) = \max_a \langle R_a, b\rangle, \tag{4.7}$$

and, therefore, if we define

$$\{\alpha_0^a\}_a = \{R_a\}_{a\in A}, \tag{4.8}$$

we have that, as desired

$$V_0(b) = \max_{\{\alpha_0^k\}_k} \langle \alpha_0^k, b \rangle. \tag{4.9}$$

As in Sec. 3.3, we turn (4.3) into an update equation:

$$V_{n+1}(b) = \max_{a \in A} \Big[\langle \alpha_0^a, b \rangle + \gamma \sum_o p(o|b,a) V_n(b^{ao}) \Big], \tag{4.10}$$

where

$$V_n(b^{ao}) = \max_{\{\alpha_n^k\}_k} \langle \alpha_n^k, b^{ao} \rangle. \tag{4.11}$$

Substituting the belief update (2.11) results in

$$V_n(b^{ao}) = \max_{\{\alpha_n^k\}_k} \Big\langle \alpha_n^k \, , \, \frac{p(o|s',a)}{p(o|b,a)} p(s'|b,a) \Big\rangle_{s'} \tag{4.12}$$

$$= \frac{1}{p(o|b,a)} \max_{\{\alpha_n^k\}_k} \Big\langle \alpha_n^k \, , \, p(o|s',a) p(s'|b,a) \Big\rangle_{s'} \tag{4.13}$$

where $\langle \cdot, \cdot \rangle_{s'}$ denotes that the expectation is defined over s'. Plugging (4.13) in (4.10) leads to

$$V_{n+1}(b) = \max_a \Big[\langle \alpha_0^a, b \rangle + \gamma \sum_o \max_{\{\alpha_n^k\}_k} \Big\langle \alpha_n^k \, , \, p(o|s',a) p(s'|b,a) \Big\rangle_{s'} \Big] \tag{4.14}$$

$$= \max_a \Big[\langle \alpha_0^a, b \rangle + \gamma \sum_o \max_{\{\alpha_n^k\}_k} \Big\langle \alpha_n^k \, , \, p(o|s',a) \int_s p(s'|s,a) b(s) \Big\rangle_{s'} \Big] \tag{4.15}$$

$$= \max_a \Big[\langle \alpha_0^a, b \rangle + \gamma \sum_o \max_{\{\alpha_n^k\}_k} \int_{s'} \alpha_n^k(s') p(o|s',a) \int_s p(s'|s,a) b(s) \Big] \tag{4.16}$$

$$= \max_a \Big[\langle \alpha_0^a, b \rangle + \gamma \sum_o \max_{\{\alpha_n^k\}_k} \int_s \big[\int_{s'} \alpha_n^k(s') p(o|s',a) p(s'|s,a) \big] b(s) \Big] \tag{4.17}$$

$$= \max_a \Big[\langle \alpha_0^a, b \rangle + \gamma \sum_o \max_{\{\alpha_n^k\}_k} \Big\langle \int_{s'} \alpha_n^k(s') p(o|s',a) p(s'|s,a) \, , \, b \Big\rangle_s \Big]. \tag{4.18}$$

At this point, analogous to (3.11), we define

$$g_{ao}^k(s) = \int_{s'} \alpha_n^k(s') p(o|s',a) p(s'|s,a), \tag{4.19}$$

and rewrite (4.18) as

$$V_{n+1}(b) = \max_a \Big[\langle \alpha_0^a, b \rangle + \gamma \sum_o \max_{\{\alpha_n^k\}_k} \langle g_{ao}^k, b \rangle \Big]. \tag{4.20}$$

Defining

$$g_{aob} = \arg\max_{\{g_{ao}^k\}_k} \langle g_{ao}^k, b \rangle, \tag{4.21}$$

we can write

$$V_{n+1}(b) = \max_a \Big[\langle \alpha_0^a, b \rangle + \gamma \sum_o \langle g_{aob}, b \rangle \Big] \tag{4.22}$$

$$= \max_a \Big\langle \alpha_0^a + \gamma \sum_o g_{aob} \, , \, b \Big\rangle. \tag{4.23}$$

We define

$$\{\alpha_{n+1}^k\}_k = \bigcup_b \Big\{ \alpha_0^a + \gamma \sum_o g_{aob} \Big\}_a, \qquad \text{for all } b \in \Delta(S). \tag{4.24}$$

Using a reasoning parallel to that of the enumeration phase of Monahan (1982)'s algorithm (see Sec. 3.4.1), we have at most $|A||\{\alpha_n^k\}|^{|O|}$ different α_{n+1}^k functions (fixing the action, we can select one of the $|\{\alpha_n^k\}|$ $g_{a,o}^k$-functions for each one of the observations).

With the above definition, we have V_{n+1} in the desired form

$$V_{n+1}(b) = \max_{\{\alpha_{n+1}^k\}_k} \langle \alpha_{n+1}^k, b \rangle, \tag{4.25}$$

and, thus, the theorem holds. □

The proof is constructive as it shows how to compute the α functions constituting V_{n+1} from those defining V_n, and as such provides a value-iteration algorithm for continuous-state POMDPs. Furthermore, we can prove that V_n is PWLC for any finite horizon as follows. From the definition of the $\langle \cdot, \cdot \rangle$ operator (4.5), for each α_n^k, $\langle \alpha_n^k, b \rangle$ is linear. As for any n the $\{\alpha_n^k\}_k$ set has a finite number of elements, the value function V_n is piecewise linear. The convexity is given by the fact that V_n is defined as the maximum of the convex (linear) functions and, thus, we obtain a convex function as a result.

4.3.2 Value iteration for continuous-state POMDPs with Gaussian models

In this section we will extend approximate POMDP value-iteration methods to continuous state spaces, with a particular focus on Perseus. As we saw in the previous section, in the case of a continuous state space the value function is still PWLC and value iteration will converge, but it requires computing integrals over the state space. We need a representation that allows analytical computation of integrals and that is also closed under Bellman backups. Perseus can be used with different representations for the beliefs, the α functions, and the transition, observation and reward models, but the selected representations have to fulfill a number of requirements.

The first requirement is that the representation used for the beliefs must be closed under the propagation through the transition model (4.2) and after the multiplication with the observation model (2.11). Next, the representation for the α functions must be closed under addition and scaling (in order to compute (4.24)), and closed for integration after multiplication with the observation and the transition model (4.19). Finally, we need a way to compute the $\langle \alpha, b \rangle$ expectation operator. For discrete-state POMDPs, the belief and the α functions are represented by vectors and the models by matrices. In this case all operations are linear algebra and produce closed results. Next, we describe a suitable representation for continuous-state POMDPs based on Gaussian functions for representing the transition, observation, and reward models as well as the beliefs.

4.3.2.1 Gaussian models for continuous-state POMDPs

Functions defined over a continuous state space (e.g., value functions and belief states) may have arbitrary forms that may not be parameterizable. In order to design feasible algorithms for continuous-state POMDPs, it is crucial to work with classes of functions that have simple parameterizations and that are closed under belief updates and Bellman backups. In this section we will focus on models based on Gaussian distributions, but note that other families of integrable functions could be used to determine the α functions in closed form[1]. Gaussian-based models provide a high degree of flexibility and are of common use in many applications, for instance in robotics (Thrun, Burgard, and Fox, 2005).

We assume that the observation model for a given observation is a Gaussian distribution (or a mixture of them) on the state space, defined non-parametrically from a set of N samples $\Omega = \{(s_i, o_i)\}$ with o_i an observation obtained at state s_i. The training set can be obtained in a supervised way (Vlassis et al., 2002) or by direct interaction with the environment (Porta and Kröse, 2004). Using these samples, the observation model can be defined as

$$p(o|s) = \frac{p(s|o)p(o)}{p(s)}, \tag{4.26}$$

where we drop the dependence on action a for convenience, and, assuming a uniform $p(s)$ in the space covered by Ω, and approximating $p(o)$ from the samples

[1] In a recent work, Poupart, Vlassis, Hoey, and Regan (2006) have shown that α functions in Bayesian reinforcement learning can be parameterized by multivariate polynomials.

in the training set we have

$$p(o|s) \approx \Big[\frac{1}{N_o}\sum_{i=1}^{N_o} \lambda_i^o \phi(s|s_i^o, \Sigma_i^o)\Big]\frac{N_o}{N} = \sum_{i=1}^{N_o} w_i^o \phi(s|s_i^o, \Sigma_i^o) \tag{4.27}$$

with s_i^o one of the N_o points in Ω with o as an associated observation, ϕ a Gaussian with mean s_i^o and covariance matrix Σ_i^o and where $w_i^o = \lambda_i^o/N$ is a weighting factor associated with that training point. The sets $\{\lambda_i^o\}_i$ and $\{\Sigma_i^o\}_i$ should be defined so that

$$p(s) = \sum_o p(s|o)p(o) = \sum_o \sum_{i=1}^{N_o} w_i^o \phi(s|s_i^o, \Sigma_i^o) \tag{4.28}$$

is (approximately) uniform in the area covered by Ω or, in other words, so that

$$\sum_o p(o|s) \approx 1. \tag{4.29}$$

As far as the transition model is concerned, we assume it is linear-Gaussian

$$p(s'|s,a) = \phi(s'|s+\delta(a), \Sigma^a). \tag{4.30}$$

with ϕ a Gaussian centered at $s+\delta(a)$ with covariance Σ^a. Nonlinear transition models can be approximated by local linearization as in the extended Kalman filter (Sorenson, 1985).

Finally, the reward can be seen as an observation with an associated scalar value. Therefore, assuming a finite set of possible rewards $R = \{r_i | i \in [1, N_R]\}$, the reward model $p(r|s,a)$ for each particular a can be represented in the same way as the observation model

$$p(r|s,a) \approx \sum_{i=1}^{N_R} w_i^r \phi(s|s_i^r, \Sigma_i^r). \tag{4.31}$$

With that, we have that

$$R_a(s) = \sum_{r\in R} r p(r|s,a) \approx \sum_{r \in R} r \sum_{i=1}^{N_R} w_i^r \phi(s|s_i^r, \Sigma_i^r), \tag{4.32}$$

which is an unnormalized Gaussian mixture.

4.3.2.2 Belief representation and updates

We will assume that belief points are represented as Gaussian mixtures of N_b components

$$b(s) = \sum_j^{N_b} w_j \phi(s|s_j, \Sigma_j), \tag{4.33}$$

with ϕ a Gaussian with mean s_j and covariance matrix Σ_j and where the mixing weights satisfy $w_j > 0$, $\sum_j w_j = 1$. In the extreme case, Gaussian mixtures with an infinite number of components would be necessary to represent a given point in the infinite-dimensional belief space of a continuous-state POMDP. However, only Gaussian mixtures with few components are needed in practical situations.

The belief update (2.11) consists of two steps and can be implemented as follows. The first step is the application of the transition model on the current belief state. This can be computed as the propagation of the Gaussians representing $b(s)$ (4.33) through the transition model (4.30) as follows:

$$\begin{aligned} p(s'|b,a) &= \int_s p(s'|s,a)b(s) && (4.34)\\ &= \sum_j w_j \phi(s|s_j + \delta(a), \Sigma_j + \Sigma^a). && (4.35)\end{aligned}$$

The second part of the belief update corrects the prediction from the transition model with the information obtained from the observation model:

$$\begin{aligned} b^{ao}(s') &\propto \Big[\sum_i w_i^o \phi(s'|s_i^o, \Sigma_i^o)\Big]\Big[\sum_j w_j \phi(s|s_j + \delta(a), \Sigma_j + \Sigma^a)\Big] && (4.36)\\ &= \sum_{i,j} w_i^o w_j \phi(s'|s_i^o, \Sigma_i^o)\phi(s|s_j + \delta(a), \Sigma_j + \Sigma^a). && (4.37)\end{aligned}$$

To compute this equation, we have to perform the product of two Gaussians. Fortunately, a closed formula is available for this operation

$$\phi(x|a, A)\phi(x|b, B) = \delta\phi(x|c, C) \qquad (4.38)$$

with

$$\begin{aligned} \delta &= \phi(a|b, A + B) = \phi(b|a, A + B), && (4.39)\\ C &= (A^{-1} + B^{-1})^{-1}, \quad c = C(A^{-1}a + B^{-1}b). && (4.40)\end{aligned}$$

Therefore, we have that

$$b^{ao}(s') \propto \sum_{i,j} w_i^o w_j \delta_{ij}^{ao} \phi(s'|s_{ij}^{ao}, \Sigma_{ij}^{ao}), \qquad (4.41)$$

with

$$\begin{aligned} \delta_{ij}^{ao} &= \phi(s_j + \delta(a)|s_i^o, \Sigma_i^o + \Sigma_j + \Sigma^a), && (4.42)\\ \Sigma_{ij}^{ao} &= ((\Sigma_i^o)^{-1} + (\Sigma_j + \Sigma^a)^{-1})^{-1}, && (4.43)\\ s_{ij}^{ao} &= \Sigma_{ij}^{ao}((\Sigma_i^o)^{-1}s_i^o + (\Sigma_j + \Sigma^a)^{-1}(s_j + \delta(a))). && (4.44)\end{aligned}$$

Finally, we can re-arrange the terms to get

$$b^{ao}(s') \propto \sum_{k}^{N_b'} w_k \phi(s'|s_k, \Sigma_k). \tag{4.45}$$

The proportionality in the definition of $b^{ao}(s')$ implies that the weights (w_k, $\forall k$) should be scaled to sum to one. The number of components $N_b' = N_b N_o$ in b^{ao} grows compared to N_b, the number of components in b. Appendix A details a Gaussian mixture condensation algorithm due to Goldberger and Roweis (2005), which can be used to bound the number of components of a Gaussian mixture while losing as little information as possible. A similar algorithm is described by Vlassis and Verbeek (2004). Next we will consider the representation of the α functions.

4.3.2.3 Representing α functions

Theorem 4.2. Assuming that the observation, transition and reward models are Gaussian-based, the functions $\alpha_n^k(s)$ can be expressed as linear combinations of Gaussian functions.

Proof. This theorem can be proved via induction. For $n = 0$, $\alpha_0^a(s) = R_a(s)$ for a fixed a and thus it is indeed an unnormalized Gaussian mixture. For $n > 0$, we assume that

$$\alpha_n^j(s') = \sum_k w_k^j \phi(s'|s_k^j, \Sigma_k^j). \tag{4.46}$$

Then, with our particular models, $g_{ao}^j(s)$ in (4.19) is the integral of three linear combinations of Gaussians

$$\begin{aligned} g_{ao}^j(s) &= \int_{s'} \Big[\sum_k w_k^j \phi(s'|s_k^j, \Sigma_k^j)\Big]\Big[\sum_l w_l^o \phi(s'|s_l^o, \Sigma_l^o)\Big]\phi(s'|s+\delta(a), \Sigma^a) && (4.47)\\ &= \int_{s'} \sum_{k,l} w_k^j w_l^o \phi(s'|s_k^j, \Sigma_k^j)\phi(s'|s_l^o, \Sigma_l^o)\phi(s'|s+\delta(a), \Sigma^a) && (4.48)\\ &= \sum_{k,l} w_k^j w_l^o \int_{s'} \phi(s'|s_k^j, \Sigma_k^j)\phi(s'|s_l^o, \Sigma_l^o)\phi(s'|s+\delta(a), \Sigma^a). && (4.49) \end{aligned}$$

As mentioned before, the product of two Gaussian functions is a scaled Gaussian. In the above case, we have to apply (4.38) twice, once for $\phi(s'|s_k^j, \Sigma_k^j)$ and $\phi(s'|s_l^o, \Sigma_l^o)$ to get $(\delta_{kl}^{jo}\phi(s'|s_1, \Sigma_1))$ and once more for $(\delta_{kl}^{jo}\phi(s'|s_1, \Sigma_1))$ and $\phi(s'|s+\delta(a), \Sigma^a)$ to get $(\delta_{kl}^{jo}\beta_{kl}^{joa}(s)\phi(s'|s, \Sigma))$. The scaling terms δ_{kl}^{jo} and $\beta_{kl}^{joa}(s)$ can be expressed as

$$\delta^{jo}_{kl} = \phi(s^o_l | s^j_k, \Sigma^j_k + \Sigma^o_l), \tag{4.50}$$

$$\beta^{joa}_{kl}(s) = \phi(s | s^{jo}_{kl} - \delta(a), \Sigma^{jo}_{kl} + \Sigma^a), \tag{4.51}$$

with

$$\Sigma^{jo}_{kl} = [(\Sigma^j_k)^{-1} + (\Sigma^o_l)^{-1}]^{-1}, \tag{4.52}$$

$$s^{jo}_{kl} = \Sigma^{jo}_{kl}[(\Sigma^j_k)^{-1} s^j_k + (\Sigma^o_l)^{-1} s^o_l]. \tag{4.53}$$

With this, we have

$$g^j_{ao}(s) = \sum_{k,l} w^j_k w^o_l \int_{s'} \delta^{jo}_{kl} \beta^{joa}_{kl}(s) \phi(s'|s, \Sigma) \tag{4.54}$$

$$= \sum_{k,l} w^j_k w^o_l \delta^{jo}_{kl} \beta^{joa}_{kl}(s) \int_{s'} \phi(s'|s, \Sigma) \tag{4.55}$$

$$= \sum_{k,l} w^j_k w^o_l \delta^{jo}_{kl} \beta^{joa}_{kl}(s). \tag{4.56}$$

Using (4.21) and (4.24), we define the elements in $\{\alpha^i_{n+1}\}$ as

$$\alpha^i_{n+1} = \alpha^a_0 + \gamma \sum_o \underset{\{g^j_{ao}\}_j}{\arg\max} \langle g^j_{ao}, b \rangle. \tag{4.57}$$

Since the result of the arg max is just one of the members of the set $\{g^j_{ao}\}_j$, all the elements involved in the definition of α^i_{n+1} are linear combinations of Gaussians and so is the final result. □

One point that deserves special consideration is the explosion of the number of components in the Gaussian mixtures defining the α functions. If C_o is the average number of components in the observation model and C_r is the average number of components in the reward model, the number of components in the α_n-functions scales with $O((|O|C_o)^n C_r)$. Again we apply the Gaussian mixture condensation procedure described in Appendix A to bound the number of components in each α functions. However, Algorithm A.1 is defined for normalized Gaussian mixtures, while the α functions are unnormalized mixtures. Therefore, we modify the procedure by normalizing the weights after taking their absolute value, which ensures that both negative and positive values are preserved. After the compression, we apply the inverse procedure to recover the original scale of the weights.

The remaining requirement on the proposed continuous-state POMDP representation is that the expectation operator $\langle \cdot, \cdot \rangle$ can be computed in closed form. As both belief and α functions are Gaussian mixtures, we compute $\langle\cdot,\cdot\rangle$ as follows:

$$\langle\alpha, b\rangle = \int_s \Big[\sum_k w_k\phi(s|s_k,\Sigma_k)\Big]\Big[\sum_l w_l\phi(s|s_l,\Sigma_l)\Big] \tag{4.58}$$

$$= \sum_{k,l} w_k w_l \int_s \phi(s|s_k,\Sigma_k)\phi(s|s_l,\Sigma_l) \tag{4.59}$$

$$= \sum_{k,l} w_k w_l \phi(s_l|s_k,\Sigma_k+\Sigma_l) \int_s \phi(s|s_{k,l},\Sigma_{k,l}) \tag{4.60}$$

$$= \sum_{k,l} w_k w_l \phi(s_l|s_k,\Sigma_k+\Sigma_l). \tag{4.61}$$

4.4 Related work

Continuous domains have received far less attention in the POMDP literature compared to traditional discrete settings. Here we will relate our work to other studies of POMDPs with continuous action or state spaces, and broaden the picture to continuous observation spaces.

4.4.1 Continuous action spaces

The literature on POMDPs with continuous actions is still relatively sparse (Thrun, 2000; Ng and Jordan, 2000; Baxter and Bartlett, 2001). In the Monte Carlo POMDP (MC-POMDP) method of Thrun (2000) real-time dynamic programming is applied on a POMDP with a continuous state and action space. In that work beliefs are represented by sets of samples drawn from the state space, while $Q(b,a)$ values are approximated by nearest-neighbor interpolation from a (growing) set of prototype values and are updated by online exploration and the use of sampling-based Bellman backups. In contrast with Perseus, the MC-POMDP method does not exploit the piecewise linearity and convexity of the value function.

Certain policy search methods tackle continuous actions, for instance Pegasus (Ng and Jordan, 2000), which estimates the value of a policy by simulating trajectories from the POMDP using a fixed random seed, and adapts its policy in order to maximize this value. Pegasus can handle continuous action spaces at the cost of a sample complexity that is polynomial in the size of the state space (Ng and Jordan, 2000, Theorem 3). Baxter and Bartlett (2001) propose a policy gradient method that searches in the space of randomized policies, and which can also handle continuous actions. The main disadvantages of policy search methods are the need to choose a particular policy class and the fact that they are prone to local optima.

4.4.2 Continuous state spaces

Also in this case only few methods exist that handle continuous state spaces directly. A common approach is to assume a discretization of the state space, which can be a poor model of the underlying system. However, when the system is linear and the reward function is quadratic, an exact solution for continuous-state POMDPs is known that can be computed in closed form (Bertsekas, 2000). While such an assumption on the reward function can be reasonable in certain control applications, it is a severe restriction for the type of AI applications we consider.

Roy (2003) has proposed compression techniques for handling POMDPs with large (discrete) state spaces, one of which compresses beliefs to two parameters: the state with maximum likelihood and the belief's entropy. Such a representation will lead to poor performance when multi-modal beliefs occur, and is only appropriate in particular applications. Recently, Brooks, Makarenko, Williams, and Durrant-Whyte (2005) have opted for a (mixture of) Gaussian functions as parametric belief representation in a continuous-state POMDP. Both methods compute an approximate value function on a grid in their low-dimensional parameter spaces, and do not use the PWLC property of the POMDP value function. In contrast, we exploit the known shape of the value function, which offers an attractive potential for generalization through the use of α functions, analogous to the effective exploitation of α vectors in discrete-state Perseus.

As mentioned before, the MC-POMDP algorithm can operate on continuous state spaces (Thrun, 2000). Value iteration is performed in the belief space, where the Bellman backup operator is approximated by sampling from the belief transition model, whereas in our case, we compute the Bellman backup operator analytically given the particular value-function representation. The MC-POMDP algorithm maintains a value function over a (growing) set of prototype beliefs, and nearest-neighbor interpolation is used to approximate the value of beliefs outside the set. This is in contrast with our Gaussian mixture representation, in which the value function achieves generalization through a set of α functions. When the value function maintained by MC-POMDP does not contain enough neighbors within a certain distance for an encountered belief, the belief is added to the value function. Perseus operates on a fixed set of beliefs, and does not require its online expansion. Furthermore, the Perseus value function is likely to generalize better over the belief space through the use of α functions.

Duff (2002) considered the problem of Bayesian reinforcement learning, in which the transition and reward model of an MDP are treated as random variables. Experience in the form of observed state transitions and received rewards is used to estimate the unknown MDP models. In contrast with straightforward exploration strategies such as ϵ-greedy (see Sec. 2.2.3), Bayesian reinforcement-learning techniques try to identify the action that will maximize long-term reward. Such an optimally exploring action might sacrifice expected immediate payoff for refining the model estimates, thus facilitating better control in the future. Duff (2002) models the Bayesian reinforcement-learning problem as a POMDP, in which the parameters of the transition model form the state of the system, and experienced transition tuples (s, a, s') are the possible observations. Such a POMDP has a continuous state space as the transition probabilities can be any real number between zero and one. A Monte Carlo algorithm is proposed for learning a (stochastic) finite-state controller for this particular class of POMDPs. The required integrals are approximated by sampling and numerical methods, whereas we assume a particular family of models for which closed-form solutions are available. Recently, Perseus has been extended to the Bayesian reinforcement-learning setting, in which the value function is represented by multivariate polynomials (Poupart et al., 2006).

4.4.3 Continuous observation spaces

Traditional POMDP methods assume discretized observation spaces as we have seen in Chapter 3, and POMDPs with continuous observation spaces have mainly been studied in model-free settings (Whitehead and Lin, 1995; Meuleau, Peshkin, Kim, and Kaelbling, 1999b; Bakker, Linåker, and Schmidhuber, 2002; Bakker, 2002). Recall that the backup of a belief point b using the forward-projected beliefs involves the following procedure:

$$\texttt{backup}(b) = \arg\max_{\{g_a^b\}_{a \in A}} b \cdot g_a^b, \tag{3.18}$$

with

$$g_a^b = \alpha_0^a + \gamma \sum_o h_{ao}^b, \tag{3.24}$$

$$\text{where } h_{ao}^b(s) = \sum_{s'} p(o|s', a) p(s'|s, a) \alpha_{ao}^b(s'), \tag{3.23}$$

$$\text{and } \alpha_{ao}^b(s') = \arg\max_{\{\alpha_n^k\}_k} b^{ao} \cdot \alpha_n^k. \tag{3.20}$$

With a continuous observation space this sum over observations becomes an integral

$$g_a^b = \alpha_0^a + \gamma \int_O \sum_{s'} p(o|s', a) p(s'|s, a) \arg\max_{\{\alpha_n^k\}_k} b^{ao} \cdot \alpha_n^k. \tag{4.62}$$

Hoey and Poupart (2005) proposed the idea that continuous observation spaces can always be discretized without any loss of information into regions corresponding to each α vector representing the value function. Intuitively, observations need to be differentiated only to the extent where they lead to different courses of actions (corresponding to different α vectors). In (4.62), all observations that lead to belief states b^{ao} with the same maximizing α vector can be aggregated together into one meta observation O_j^{ba} defined as follows:

$$O_j^{ba} = \{o | \alpha_n^{O_j^{ba}} = \arg\max_{\{\alpha_n^k\}_k} b^{ao} \cdot \alpha_n^k\}. \tag{4.63}$$

Using O_j^{ba}, we can rewrite (4.62) as follows:

$$g_a^b = \alpha_0^a + \gamma \sum_{s'} p(s'|s,a) \sum_j p(O_j^{ba}|s',a) \alpha_n^{O_j^{ba}}. \tag{4.64}$$

Hoey and Poupart (2005) propose to simplify the above quantity by accumulating probability masses over observations in O_j^{ba}, i.e., defining $p(O_j^{ba}|s',a) = \int_{o \in O_j^{ba}} p(o|s',a)$, and then approximating the discrete $p(O_j^{ba}|s',a)$ by sampling observations from each state s' (given the action a). Observe that in the continuous-observation case, the dynamic partitioning of the observation space is performed for the particular belief point b that we would like to backup. Therefore, it cannot be used for enumeration-style algorithms such as Monahan's algorithm (Sec. 3.4.1) or Incremental Pruning (Sec. 3.4.3), which do not use point-based backups. However, it can be integrated in point-based methods such as PERSEUS and PBVI (Sec. 3.5).

4.5 Experimental results

We report on experiments in two types of domains: we test PERSEUS in domains with continuous action spaces followed by an experiment with continuous states. In all reported experiments the discount factor γ is set to 0.95.

4.5.1 Continuous action spaces

We applied PERSEUS in two domains with continuous action spaces: an agent equipped with proximity sensors moving at a continuous heading and distance, and a navigation task involving a mobile robot with omnidirectional vision in a perceptually aliased office environment (Spaan and Vlassis, 2005a,b).

4.5.1.1 Continuous Navigation

We first tested our approach on a navigation task in a simulated environment, in which an agent can move at a continuous heading and distance. The Continuous Navigation environment represents a 20 × 10m hallway which is highly perceptually aliased (see Fig. 4.1). The agent inhabiting the hallway is equipped

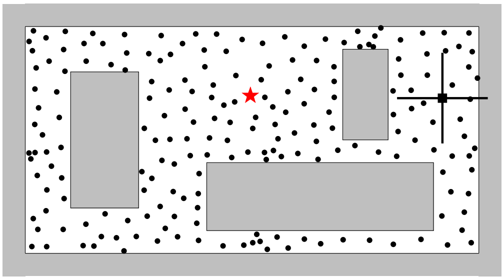


Figure 4.1: State space of the Continuous Navigation problem, where points indicate the states, and ★ depicts the goal state. The black square represents the agent, the four beams indicate the range of its proximity sensors.

with four proximity sensors, each observing one compass direction. We assume that a proximity sensor can only detect whether there is a wall within its range of 2m or not, resulting in a total number of 16 possible sensor readings. The agent's sensor system is noisy: with 0.9 probability the correct wall configuration is observed, otherwise one of the other 15 observations is returned with equal probability. The task is to reach a goal location located in an open area where there are no walls near enough for the agent to detect. The agent is initialized at a random state in the environment, and it should learn what movement actions to take in order to reach the goal as fast as possible.

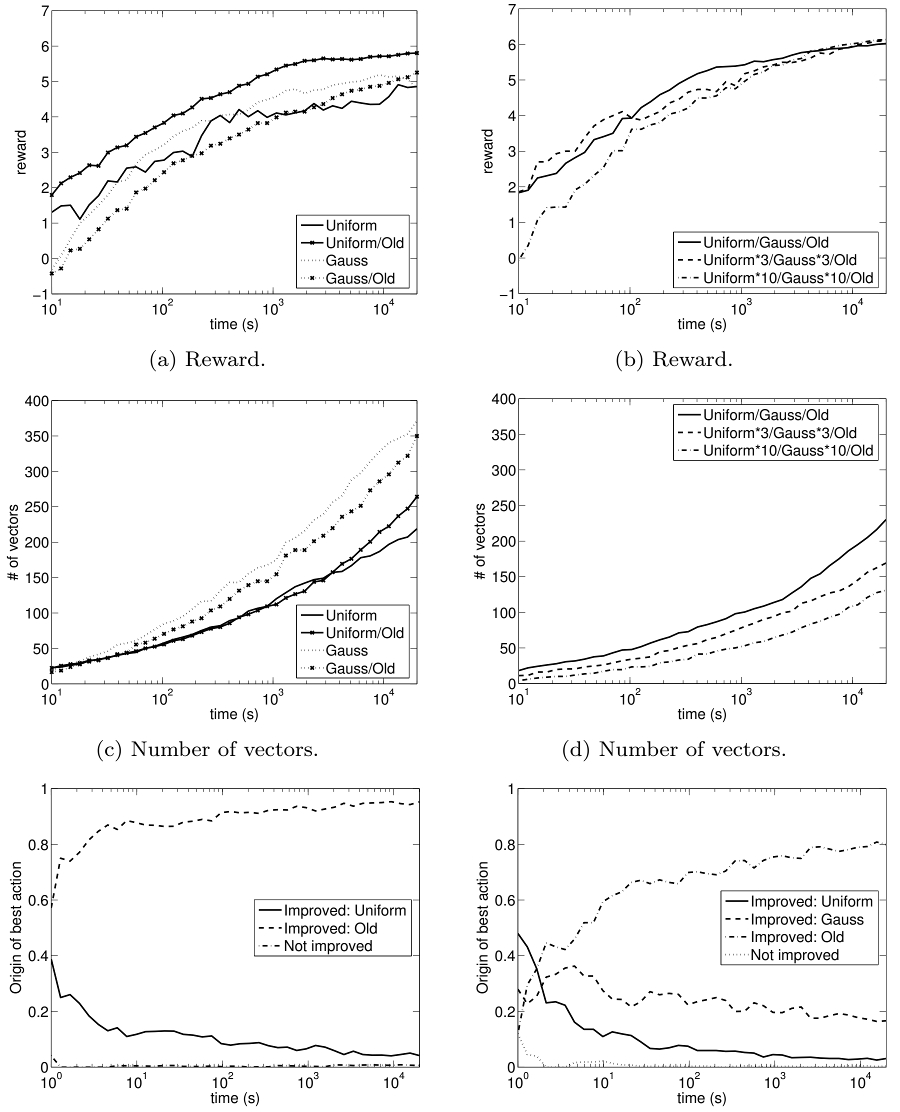
In this section we assume a finite and discrete state space S (the set of all possible locations of the agent) and we performed a simple k-means clustering on a random subset of all possible locations, resulting in a grid of 200 locations depicted in Fig. 4.1. The agent's actions are defined by two parameters: the heading θ to which the agent turns and the distance d it intends to move in this direction. Executing an action transports it according to a Gaussian distribution centered on the expected resulting position, which is defined as its current (x, y) position translated d meter in the direction θ. The standard deviation of the Gaussian transition model is $0.25d\,\mathbf{I}$, which means the further the agent wants to travel, the more uncertainty there will be regarding its resulting position. The distance parameter d is limited to the interval $[0, 2]$m and the heading θ ranges on $[0, 2\pi]$. Each movement is penalized with a reward of -0.1 per step and the reward obtainable at the goal location is 10.

To test the feasibility of Perseus in continuous action spaces, i.e., whether it can compute successful policies by sampling actions at random, we experimented with a number of different sampling schemes for the `backup′` operator. Each scheme is defined by the makeup of $A'_b = \{A^{\mathcal{U}}, A_b^{\mathcal{N}}, A_b^{old}\}$, which is composed of samples from three distributions: $A^{\mathcal{U}}$, uniformly at random; $A_b^{\mathcal{N}}$, a Gaussian distribution centered on the best known action $a(\alpha_n^b)$ for b so far, with standard deviation $\sigma_\theta = \frac{\pi}{5}$ for θ and $\sigma_d = 0.1$ for d; and A_b^{old}, a Dirac distribution on the best known action. We will describe A'_b by the number of samples from each distribution $\{|A^{\mathcal{U}}|, |A^{\mathcal{N}}_b|, |A^{old}_b|\}$. We tested the following schemes: sampling a single action uniformly at random $\{1, 0, 0\}$, or from a Gaussian distribution on $a(\alpha^b_n)$ $\{0, 1, 0\}$; adding $a(\alpha^b_n)$ to both schemes resulting in $\{1, 0, 1\}$ and $\{0, 1, 1\}$; and $\{k, k, 1\}$, sampling k actions from the uniform and Gaussian distributions and including the old action. The latter scheme explores the option of sampling more than one action from a particular distribution, and we tested $k = \{1, 3, 10\}$. The option to try the best known ('old') action for the particular b is relatively cheap as we can cache its transition, observation, and reward model the first time it is chosen (at a previous backup stage).

In this problem we used a set B of 10,000 belief points. To evaluate the control quality of the computed value functions we collected rewards by sampling 10 trajectories from 100 random starting locations at particular time intervals, while following the policy computed so far. Each trajectory was stopped after a maximum of 100 steps (if the agent had not reached the goal by then), and the collected reward was properly discounted. All results are averaged over 10 runs of Perseus with a different random seed and are computed in Matlab on an Intel Xeon 3.4GHz.

Fig. 4.2 shows the results for each of the sampling schemes mentioned above. The top row displays the control quality as indicated by the average discounted reward. In Fig. 4.2(a) we can see that just sampling a single action uniformly at random $\{1, 0, 0\}$ already gives good performance, while extending A'_b to include the best known action $\{1, 0, 1\}$ improves control quality. The Gaussian sampling schemes $\{0, 1, 0\}$ and $\{0, 1, 1\}$ learn slower as they can take only small steps in action space. An additional disadvantage of Gaussian sampling is the need for the user to specify the standard deviation. Fig. 4.2(b) depicts the control quality of the schemes in which we sample from three distributions $\{k, k, 1\}$, for different values of k. The figure shows that all tested variations reach similar control quality, but trying more actions for a particular b can slow down learning. However, when looking at the size of the value function (Fig. 4.2(c)–(d)), we see that for $k = 10$ the resulting value function is smaller than for any other scheme tested. It appears in this experiment that sampling more actions increases the chance of finding a high quality action that generalizes well (so fewer vectors are eventually needed to reach the same control quality), but at a higher computational cost per backup stage. Note that for all tested schemes the number of vectors in the value function remains two orders of magnitude lower than the size of B (10,000 belief points), confirming the efficient behavior of the Perseus randomized backup scheme.

To obtain more insight in the effect of sampling from different distributions in A'_b, we computed the relative frequency of occurrence of the maximizing action in $A^{\mathcal{U}}$, $A^{\mathcal{N}}_b$, and A^{old}_b. When executing a `backup'` we check whether the vector computed using the returned action actually improves $V(b)$, and if so, we record whether this action originated from $A^{\mathcal{U}}$, $A^{\mathcal{N}}_b$, or A^{old}_b. For every backup



(a) Reward.

(b) Reward.

(c) Number of vectors.

(d) Number of vectors.

(e) Origin of maximizing action.

(f) Origin of maximizing action.

Figure 4.2: Perseus results on the Continuous Navigation problem, averaged over 10 runs. The left column shows the performance of $\{|A^{\mathcal{U}}|, |A_b^{\mathcal{N}}|, |A_b^{old}|\} = \{\{1, 0, 0\}, \{1, 0, 1\}, \{0, 1, 0\}, \{0, 1, 1\}\}$, and the right column displays $\{k, k, 1\}$ for $k = \{1, 3, 10\}$. The top row figures display the average discounted reward obtained vs. computation time, the figures in the middle row show the value-function size, and the bottom row details the origin of the maximizing vector (see main text).

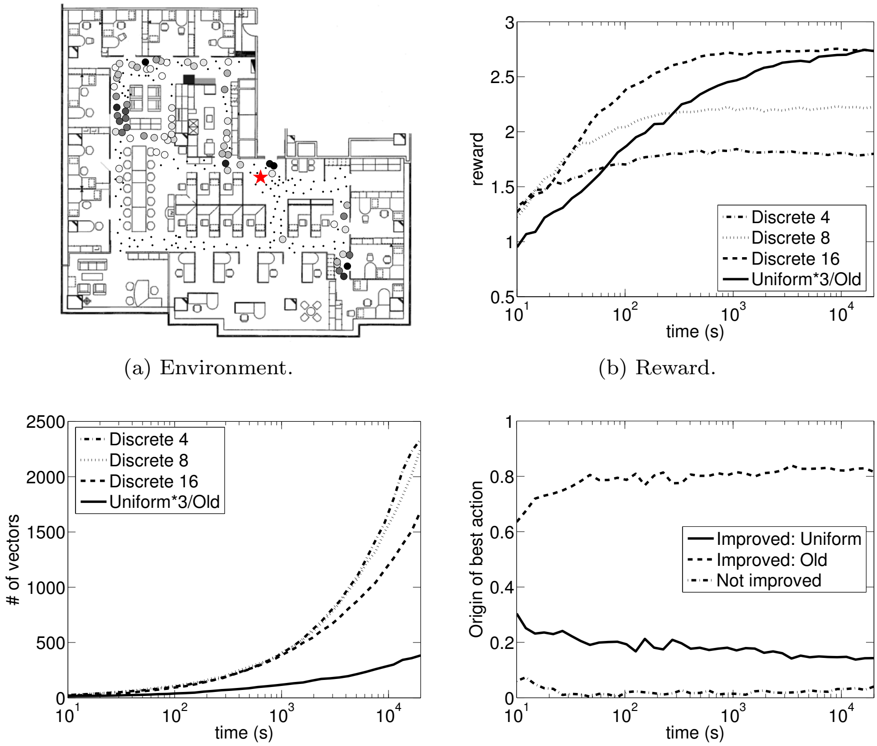
stage we normalize these counts with respect to the total number of backups in that backup stage (including those that did not improve $V(b)$). The resulting frequencies are plotted on the bottom row of Fig. 4.2 for two sampling schemes: sampling uniform and old $\{1, 0, 1\}$ (Fig. 4.2(e)) and sampling one action from all three distributions $\{1, 1, 1\}$ (Fig. 4.2(f)). We can see that over time the relative frequency of the best known action grows ("Improved: Old"), while the number of instances in which none of the sampled actions improves $V(b)$ drops to almost zero ("Not improved"). The frequencies of actions sampled from an uniform or Gaussian distribution ("Improved: Uniform" resp. "Improved: Gauss") resulting in the best action in A'_b (and improving $V(b)$) also drop. These observations confirm the intuition that by sampling actions at random Perseus can effectively explore the action space (which is advantageous at early backup stages), while as time progresses the algorithm seems to be able to exploit the actions that turn out to be useful.

4.5.1.2 Arbitrary Heading Navigation

To evaluate Perseus with continuous actions on a more realistic problem and compare with discretized action spaces we also include the cTRC domain. In this problem (adapted from Spaan and Vlassis, 2005b, see Sec. 3.8.2) a mobile robot with omnidirectional vision has to navigate in a highly perceptually aliased office environment (see Fig. 4.3(a)). We use the MEMORABLE[1] robot database that contains a set of approximately 8000 panoramic images collected manually by driving the robot around in a 17×17 meters office environment. The robot can decide to move 5m in an arbitrary direction, i.e., its actions are parameterized by its heading ranging on $[0, 2\pi]$. We applied the same technique as in the Continuous Navigation domain to grid our state space in 200 states (Fig. 4.3(a)) and assume a Gaussian error on the resulting position. For our observation model we compressed the images with PCA and applied k-means clustering to create 10 three-dimensional prototype feature vectors $\{o_1, \dots, o_{10}\}$. Fig. 4.3(a) shows the inverse observation model $p(s|o_k)$ for one particular observation. The task is to reach a certain goal state at which a reward of 10 can be obtained; each action yields a reward of -0.1. The belief set B contained 10,000 points.

We compared the continuous-action extension of Perseus to three discretized versions of this problem, in which we applied regular Perseus to a fixed discrete action set of 4, 8 or 16 headings with equal separation (offset with a random angle to prevent any bias). Fig. 4.3(b)-(d) display results for Perseus with $\{|A^{\mathcal{U}}|, |A_b^{\mathcal{N}}|, |A_b^{old}|\} = \{3, 0, 1\}$ (other schemes turned out to give similar results), and the three discrete action spaces. For discrete action spaces, when the action space A is finite and relatively small (and S is a finite set), one can cache in advance a (sparse) tabular representation of the transition, observation,

[1] The MEMORABLE database has been provided by the Tsukuba Research Center in Japan, for the Real World Computing project.



(a) Environment.

(b) Reward.

(c) Number of vectors.

(d) Origin of best action.

Figure 4.3: cTRC Domain. (a) Environment of the cTRC problem, where points indicate the states, and ★ depicts the goal state. The induced $p(s|o_k)$ for a particular o_k is also shown: the darker the disk, the higher the probability. (b)-(d) Performance of Perseus, averaged over 10 runs.

and reward models for all $a \in A, s \in S$. Such a cached tabular representation allows for an optimized implementation of the `backup` operator. In contrast, when sampling actions from a continuous action space A, one has to compute the above models 'on the fly' for each sampled action $a \in A'_b$, which requires an algorithm (a parameterized model family) that can generate the transition, observation and reward models for any action that is given as input. Such generated models can be cached for later use in case the same action is considered again in future iterations.

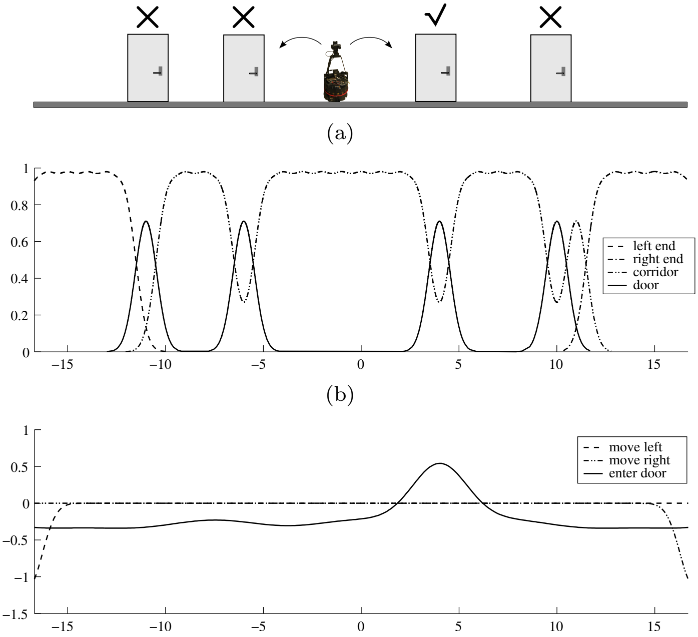
Fig. 4.3(b) shows that sampling from a continuous A results in the same control quality as in the discrete 16 version, but it needs more time to reach it (as the `backup′` requires to generate transition, observation and reward models on the fly). As the discrete cases benefit from an optimized implementation, the continuous-action scheme needs some computation time to match performance or outperform them. However, when employing the continuous scheme, Perseus exploits the ability to move at an arbitrary heading to find a better policy than the discrete 4 and 8 cases. We see that providing the robot with a more fine-grained action space leads to better control quality, and in this problem a discretization of 16 headings appears to be fine-grained enough for good control performance. Fig. 4.3(c) plots the number of vectors in the value function for each scheme, where we see that for reaching the same control quality the continuous and discrete 16 version need a similar amount of vectors. Fig. 4.3(d) shows the relative frequency of occurrence of the maximizing action in $A^{\mathcal{U}}$ or A_b^{old}, as detailed in Sec. 4.5.1.1. As in Fig. 4.2(e)–(f) we see that over time the best known action is exploited, while the frequency of instances in which no sampled action improves the value of b is diminished to near zero.

4.5.2 Continuous state spaces

To demonstrate the viability of Perseus in continuous state spaces we carried out an experiment in a simulated robotic domain. In this so-called Corridor problem (see Fig. 4.4(a)) a robot is moving in a corridor with four doors. The robot can detect when it is in front of a door and when it is at the left or right end of the corridor. In any other situation, the robot just detects that it is in a corridor (see Fig. 4.4(b)). The robot can move 2 units to the left or to the right (with $\Sigma^a = 0.05$) and can try to enter a door at any point (even when not in front of a door). The target for the robot is to locate the second door from the right and to enter it. The robot only gets positive reward when it enters the target door (see Fig. 4.4(c)). When the robot tries to move further than the end of the corridor (either at the right or at the left) or when it tries to enter the door at a wrong position it receives a negative reward.

The set of beliefs B used by Perseus contains 1000 unique belief points. Those Gaussian mixtures are collected using random walks departing from a belief including 4 components that approximate a uniform distribution on the whole corridor. The trajectories of the robot along the corridor are organized in episodes, in which the robot executes actions until it tries to enter a door or until it executes 25 (movement) actions. We use Algorithm A.1 to compress beliefs to 4 components (i.e., the number of components of the initial belief) and compress α functions in such a way to ensure that they never have more components than the number used to represent the reward function (11 components).

Fig. 4.5 shows the averaged results obtained after 10 Perseus runs on this problem. Fig. 4.5(a) shows that the value computed as $\sum_b V(b)$ converges. Fig. 4.5(b) shows the expected discounted reward averaged for 100 episodes with the policy available at the corresponding time step. The plot indicates that the robot successfully learns to find out its position and to distinguish


(a)

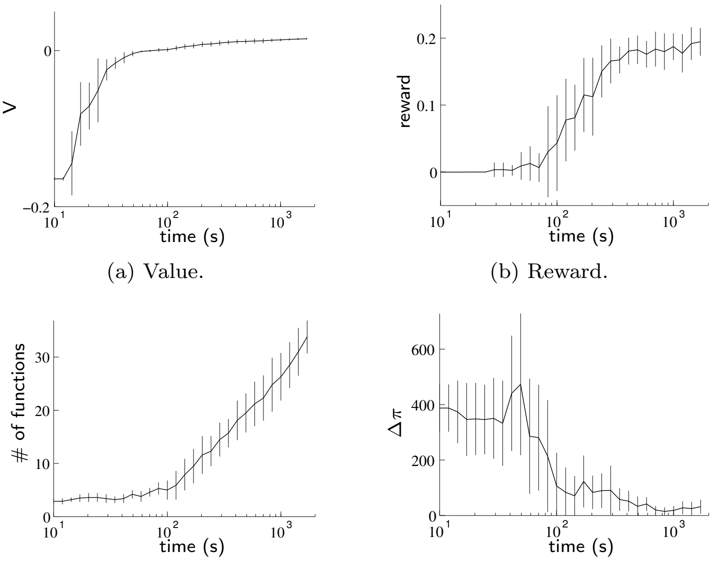
(b)

(c)

Figure 4.4: (a) A pictorial representation of the Corridor problem, (b) the corresponding observation model and (c) the reward model.

between the four doors. Fig. 4.5(c) show the number of α functions used to represent the value function. We can see that the number of α functions used increases, but is far below 1000, the maximum possible number of α functions (in the extreme case we would use a different α function for each point in B). Finally Fig. 4.5(d) we show the number of changes in the policy from one backup stage to the next one. The changes in the policy are computed as the number of elements in B with a different action compared to the previous value function. The number of policy changes drop to close to zero, indicating convergence with respect to the particular B.

Following the computed policy the robot moves to one of the ends of the corridor to determine its position and then towards the correct door to enter it. Fig. 4.6(a) through (i) show how the robot's belief evolves over time, starting at a uniform belief and ending up with a peaked belief centered at the target door. Finally, Fig. 4.6(j) plots the value for beliefs with only one component parameterized by the mean and covariance of this component. We can see that, as the uncertainty about the position of the robot grows (i.e., as the covariance is larger) the value of the corresponding belief decreases. The colors in the figure



(a) Value.

(b) Reward.

(c) No. of α functions.

(d) Policy changes.

Figure 4.5: Results for the Corridor domain.

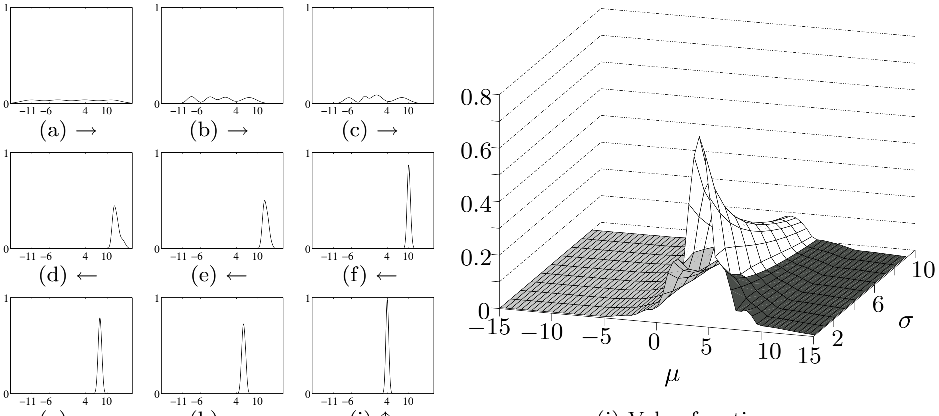
correspond the the different actions: light-gray for moving to the right, white for entering the door, and dark-gray for moving to the left.

Observe that the advantage of using a continuous state space is that we obtain a scale-invariant solution. If we have to solve the same problem in a longer corridor, we can just scale the Gaussians used in the problem definition and we will obtain the solution with the same cost as we have now. The only difference is that more actions would be needed in each episode to reach the correct door. When discretizing the environment, the granularity has to be in accordance with the size of the actions taken by the robot (± 2 left/right) and, thus, the number of states and, consequently, the cost of the planning increases as the environment grows.

4.6 Discussion

In this chapter we considered approximate planning for POMDPs in which the state or action space are continuous. We focused on Perseus, our randomized point-based POMDP planner introduced in Chapter 3, but our extensions can also be applied to other approximate POMDP solution methods.

We extended Perseus to compute plans for agents which have a very large



(a) → (b) → (c) →

(d) ← (e) ← (f) ←

(g) ← (h) ← (i) ↑

(j) Value function.

Figure 4.6: Corridor domain. (a)-(i) Evolution of the belief when following the computed policy. The arrows under the snapshots represent the actions: → for moving right, ← for moving left and ↑ for entering the door. On the x-axis the four door locations are indicated. (j) Value function for single-component beliefs as a function of the mean μ and the covariance σ.

or continuous set of actions at their disposal, by sampling actions from the action space. The concept of sampling a belief b and at the same time a set of actions A'_b, and keeping the α vector which improves the value of b most is consistent with the Perseus backup stage, as we only require that the value of b does not decrease over successive value-function approximations. If none of the sampled actions in A'_b improves $V_n(b)$ we keep the current vector α_n^b of b in V_n. The action a_{old} associated with α_n^b can give clues as to what kind of action is appropriate for b. In particular, we can include a_{old} in A'_b, or include actions sampled in the neighborhood of a_{old}. On the other hand, we can explore the action space at the same time by including actions in A'_b that are sampled from A uniformly at random. In our experiments in Sec. 4.5.1 we have investigated the effect of various sampling schemes, and we have seen that in the early backup stages exploring actions are selected, but over time actions that turn out to be useful are exploited.

Regarding POMDPs with continuous states, we demonstrated the piecewise linearity and convexity of value functions defined over infinite-dimensional belief states induced by continuous states. As the continuous-state Bellman backups are isotonic and contracting, we can adapt value iteration to continuous-state POMDPs. In particular, we extended Perseus with linear combinations of Gaussians for belief states. These are expressive representations that are closed under Bellman backups and belief updates. We showed how our method can be successfully applied to a simple robot navigation task.

In this chapter we have shown how Perseus can be extended to continuous action and state spaces. However, several other notable extensions to Perseus have been recently proposed in the POMDP literature since its introduction. Poupart (2005) and Hoey and Poupart (2005) have extended Perseus to very large or continuous observation spaces, as discussed in detail in Sec. 4.4.3, and applied it to a real-world task for assisting persons with dementia (Boger, Poupart, Hoey, Boutilier, Fernie, and Mihailidis, 2005). Furthermore, they extended Perseus to compute policies compactly represented as algebraic decisions diagrams (ADDs) (Hoey et al., 1999; Hansen and Feng, 2000), allowing them to approximately solve a POMDP model with 50 million states. Perseus has also been extended to two reinforcement-learning settings. Shani, Brafman, and Shimony (2005a) learn a POMDP model of the environment while at the same time an online, incremental version of Perseus is used to compute a policy. Poupart et al. (2006) proposed an extension to Perseus for effective online learning that is computationally efficient while minimizing the amount of exploration, by modeling the reinforcement-learning problem as a POMDP over the MDP parameters. James, Wessling, and Vlassis (2006) extended Perseus to predictive state representations, a recent alternative framework for modeling partially observable and stochastic environments (Littman, Sutton, and Singh, 2002; Singh, James, and Rudary, 2004).

A promising line of future research is to combine the three continuous extensions, resulting in a planner for fully continuous domains (with discretized time). Combining the continuous action and state extensions presented in this chapter should pose no significant problem, as they are rather orthogonal. As Hoey and Poupart (2005) already noted, integrating their dynamic partitioning of the observation space for point-based backups (see Sec. 4.4.3) in a point-based planner for continuous states and actions should be possible. In particular, the extension to continuous actions and observations both employ particular sampling strategies that exploit the properties of (continuous-state) POMDPs and Perseus.

Chapter 5

Planning for teams of agents

In Chapters 3 and 4 we considered the problem of planning under uncertainty for single agents. We covered the traditional discrete POMDP setting and explored extensions to continuous POMDP domains. In this chapter, we will focus on planning for cooperative multiagent systems.

5.1 Introduction

In this chapter we will consider the problem of planning for cooperative multiagent systems, i.e., for teams of agents. A team of agents is defined as a group of agents that inhabit the same environment, and whose performance is not judged by their individual behavior, but by the joint behavior of all agents. Or, in MDP terminology, an agent's reward function is defined on the states and actions of all agents, and the reward function and optimality criterion are identical for all agents. Planning for a team of agents is significantly harder than planning for a single agent, as the performance of an agent depends on the behavior of its teammates, and vice versa.

A fundamental question is how to approach the multiagent optimization problem. One approach would be to model the whole team as a single centralized agent, who chooses its actions from the joint-action space, and its action would dictate each agent's individual action. However, the problem description, i.e., the state, action, and observation spaces, grows exponentially in the number of agents, severely limiting the scalability of such centralized approaches. However, one potential application area is the control of a few robots which have to perform a task requiring tightly coupled coordination under physical constraints, for instance two robots who need to transport a large beam across rough terrain (Huntsberger et al., 2003).

On the other extreme, one can also compute each agent's policy independently, reducing the complexity from exponential to linear in the team size, but with a potentially detrimental effect on team performance, as the agents do not optimize their joint performance. However, it might suffice for multiagent tasks with few interdependencies and a fairly additive joint reward function, i.e., where the joint reward is the sum of individual rewards (Tan, 1993; Sen, Sekaran, and Hale, 1994). We will focus on multiagent tasks that can be positioned in between both extremes, tasks in which explicit cooperation is required at some stages, but at other points in time the agents can afford to act rather independently.

We will adopt decentralized partially observable Markov decision processes (DEC-POMDPs) as our planning paradigm (Bernstein et al., 2002). DEC-POMDPs form a general framework for planning for groups of cooperating agents that inhabit a stochastic and partially observable environment. Unfortunately, computing optimal plans in a DEC-POMDP has been shown to be intractable (NEXP-complete, Bernstein et al., 2002, see Sec. 2.5), and approximate algorithms for specific subclasses have been proposed. Many of these algorithms rely on an (approximate) solution of the centralized planning problem (i.e., treating the whole team as a single agent). We take a more decentralized approach, in which each agent only reasons over its own local state and some uncontrollable state features, which are shared by all team members (Spaan, Gordon, and Vlassis, 2006). In contrast to other approaches, we model communication as an integral part of the agent's reasoning, in which the meaning of a message is directly encoded in the policy of the communicating agent. We explore iterative methods for approximately solving such models, and we conclude with some encouraging preliminary experimental results.

5.2 Decentralized planning for teams of communicating agents

Single-agent planning under uncertainty has been formalized in the partially observable Markov decision process (POMDP) framework. A POMDP defines stochastic models to capture the uncertain effects of actions and the fact that sensors are imperfect and have a limited scope. In particular, the transition model governs how the system's state is influenced by actions, and the observation model stochastically relates observations to states. Solving a POMDP exactly is intractable, but recent years have seen much progress in the development of approximate methods, as we discussed in Chapter 3.

Planning for a team of agents without explicit communication proves to be significantly harder than planning for just a single agent. In order for the team to act optimally, each agent has to consider not only the consequences of its own action, but also consider the actions its teammates will execute. As each agent does not know what the other agents observe, it will be hard to predict their actions. Decentralized POMDPs (DEC-POMDPs) generalize the POMDP framework to cooperative multiagent settings, but solving finite-horizon DEC-POMDPs has been proven to be NEXP-complete (Bernstein et al., 2002), a considerable jump in complexity from the single-agent case. Besides brute-force enumeration and evaluation of all possible policy tuples, only one algorithm is known for computing optimal solutions (Hansen et al., 2004), which in practice is limited to very small problems. As the general setting is very hard to solve optimally, research has focused on heuristic methods, which often also make additional assumptions on the planning domain (Xuan et al., 2001; Goldman and Zilberstein, 2003; Becker et al., 2004; Guo and Lesser, 2005; Paquet, Tobin, and Chaib-draa, 2005).

Allowing agents to communicate with each other, possibly modeled outside the DEC-POMDP, can increase team performance. Communication mitigates uncertainty: a message from a teammate can for instance provide information about the state of the system, or about the teammate's intentions. In particular, if one assumes that the agents have unlimited free access to a noise-free communication channel, the multiagent planning problem reduces to a single-agent one. However, the resulting problem description grows exponentially in the number of agents. Therefore, we choose to tackle the problem in a decentralized manner, in which each agent considers only its local state plus some shared uncontrollable state features. Furthermore, we consider communication as an integral part of the system, and model it directly in the DEC-POMDP, in contrast with approaches in which communication decisions are governed by a separate algorithm (Goldman and Zilberstein, 2003; Emery-Montemerlo et al., 2004; Nair, Tambe, Roth, and Yokoo, 2004; Roth et al., 2005).

We propose a decentralized model for tackling cooperative multiagent planning problems, which incorporates a communication channel. In contrast to most work on distributed POMDPs, we do not define an explicit communication language, but instead treat the semantics of communication as part of the optimization problem. We will present an iterative method for computing a joint policy for the team in a decentralized fashion. Given a set of fixed policies for all agents but agent i, we convert the DEC-POMDP to a POMDP from agent i's perspective which incorporates the expected contribution to the joint team reward of the other agents' policies. We will propose a heuristic method for computing a communication policy, but first we will review the literature on methods for (approximately) solving general or simplified DEC-POMDPs.

5.2.1 Solving decentralized POMDPs

Hansen et al. (2004) present a dynamic-programming operator for finite-horizon partially observable stochastic games (POSGs), which allows for incremental construction of policies: starting from a solution for horizon t it computes a solution for horizon $t+1$. When the POSG is a DEC-POMDP, i.e., the agents share the same reward function, the algorithm will result in a set of policies optimal for a certain planning horizon. A belief is maintained over the cross-product of the state of system and the future conditional plans, or policy trees, of all other agents. However, the set of possible policy trees grows doubly exponentially in the planning horizon, which renders a brute-force approach infeasible in practice. To combat the growth, at every iteration (very weakly) dominated policy trees are pruned. A policy tree for agent i is dominated if its removal does not decrease the value of any belief for agent i. However, even with pruning at every iteration the algorithm can solve only very small problems before running out of memory.

As solving a DEC-POMDP is intractable in the general case, methods have been proposed for approximately solving more restricted DEC-POMDP variations. Simplified models that have been studied include extra assumptions regarding the observation model, in particular that an agent's observations are independent from those of its teammates (Goldman and Zilberstein, 2003) or that each agent can observe its own local state with full certainty, which reduces the problem to a decentralized MDP (Xuan et al., 2001; Becker et al., 2004; Guo and Lesser, 2005). The latter assumption requires a factored state representation in which the state space is the cross-product of each agent's local state space and optionally a set of shared external state features which each agent can observe but not influence (examples include time or any other external information relevant to the team's task). Given a factored state space one can further assume that the agents' transitions are independent (Xuan et al., 2001; Becker et al., 2004; Paquet et al., 2005), or impose a particular joint reward structure (Becker et al., 2004). Apart from restricting the model one can also restrict the complexity of the policy space, for instance by only searching in the space of finite-state controllers of a particular size (Bernstein, Hansen, and Zilberstein, 2005; Szer and Charpillet, 2005; Amato, Bernstein, and Zilberstein, 2006). Another way of computing a locally optimal solution is to optimize one agent's policy while keeping the other policies fixed, and iterate over all agents until convergence (Nair et al., 2003; Emery-Montemerlo et al., 2004; Bernstein et al., 2005).

A different dimension to tackle decentralized POMDP models is to assume that the agents have a way to communicate with each other (which is not modeled in the DEC-POMDP). As we noted above, if the agents in a team are allowed to communicate for free and without limitations, then each agent can broadcast at every time step its perceived observation to all of its teammates. As a result every agent knows the joint observation vector, which it can use to update the joint belief of the team. Every agent in the team selects the next joint action based on the joint belief, and each agent executes its action according to the joint action. As such, free communication reduces the distributed control problem to a centralized, single-agent one, which can be solved using standard POMDP solution techniques. However, in reality communication is often not free or unlimited, and modeling a team of agents in this way might not be desirable. Nevertheless, several methods treat the multiagent system as if communication were truly free, solve the large centralized POMDP (exactly or approximately), and execute the resulting joint policy in a distributed fashion while imposing communication constraints (Emery-Montemerlo et al., 2004; Nair et al., 2004; Roth et al., 2005). While executing the policy, a limited form of communication is used to attempt to preserve coherent behavior of the team. In these models, the communication decisions are not part of the centralized policy, but are made by a separate algorithm, which for instance monitors the uncertainty regarding the joint belief. Detailed studies of communication issues arising in DEC-POMDP settings are available (Pynadath and Tambe, 2002; Goldman and Zilberstein, 2004).

In contrast, Xuan et al. (2001) propose a model that incorporates the communication decision directly in the agent's policy. It adds a communication sub-stage to the decision process, in which an agent decides whether it will communicate with its teammates. In this way an agent can explicitly reason about communication, instead of relying on an independent instrument to handle communication issues. Extending this framework, Goldman and Zilberstein (2003) present a formal model for decentralized control with communication decisions based on the DEC-POMDP model. In such decentralized models agents reason over their local state, instead of considering the state of all teammates. A decentralized model seems more appropriate for agents who cannot observe each other's state nor have free and unrestricted communication at their disposal. Decentralized models also do not suffer from the exponential growth in the size of the centralized state space when the number of agents increases. Limited communication abilities, however, can be exploited to improve the team's performance, by allowing agents to share information at a certain cost.

5.2.2 Proposed model

Goldman and Zilberstein (2003, 2004) proposed a framework for modeling cooperative multiagent systems as a decentralized POMDP with a global reward function. The agents reason explicitly over their communication decisions, but the observations of each agent are independent, and hence, incoming messages are not represented in an agent's observation model. Instead, the agents store the full history of messages they receive. Messages are exchanged instantaneously, without any delays. We propose a different decentralized model which draws on the work of Goldman and Zilberstein (2003, 2004), but in which a communication channel is established by allowing agents to send messages as part of their action vector, and messages are received in the next time step as part of the recipients' observation vectors. In contrast to most work on distributed POMDPs, we do not manually define a communication language, but treat the semantics of communication as part of the optimization problem.

We propose the following model (Spaan et al., 2006), which is based on the DEC-POMDP model (see Sec. 2.4.2):

- $I = \{1, \dots, m\}$ is a set of m agents.

- S is a finite set of states, and factors $m+1$ components: $S = S_0 \times S_1 \times \cdots \times S_m$, where S_i is the set of state features relevant to agent i and S_0 the set of external state features, which are shared by all agents but which cannot be controlled. A local state of an agent i is defined as $\bar{s}_i = (s_0, s_i)$, where $s_0 \in S_0, s_i \in S_i$.

- Σ_i is a finite set of possible messages, and $\sigma_i \in \Sigma_i$ denotes a message sent by agent i. Not sending any message is defined as sending the empty message $\sigma_i = \emptyset$.

- $A_i = A_i^d \times A_i^\sigma$ is the action set for agent i, where A_i^d is the set of domain-level actions and A_i^σ the set of communication acts.

- $O_i = O_i^d \times O_i^\sigma$ is the observation set for agent i, where O_i^d is the set of domain-level observations the agent receives through its sensors, and $O_i^\sigma = \Sigma_1 \times \cdots \times \Sigma_{i-1} \times \Sigma_{i+1} \times \cdots \times \Sigma_m$ the set of possible messages the agent can receive from any of its teammates.

- The joint transition model $p(s'|s, \bar{a})$ depends only on the domain-level component of each agent's action (sending messages does not influence the state of the environment). We assume that the agents' transitions are independent, allowing us to factor the joint transition model:

$$p(s'|s, \bar{a}) = p(s_0'|s_0) \prod_{i=1 \dots m} p(s_i'|s_i, a_i^d), \tag{5.1}$$

  where $s_0', s_0 \in S_0$, $s_i', s_i \in S_i$, and $a_i^d \in A_i^d$.

- The local observation model $p(o_i|\bar{s}_i, a_i)$ of agent i specifies the probability of receiving observation $o_i = (o_i^d, o_i^\sigma)$ after executing action a_i and ending up in $\bar{s}_i$. The domain-level component is defined by the joint observation model $p(\bar{o}|s, \bar{a})$, i.e., the probability that after taking joint action $\bar{a}$ and ending up in state s results in joint observation $\bar{o} \in O_1^d \times \cdots \times O_m^d$. The communication component is defined by the policy of the other agents.

- We define $p(o_i^\sigma|\bar{\sigma})$ as the probability that a particular message vector $\bar{\sigma} = (\sigma_1, \dots, \sigma_{i-1}, \sigma_{i+1}, \dots, \sigma_m)$ will be interpreted as observation o_i^σ of agent i, which can be used to model noise in the communication channel.

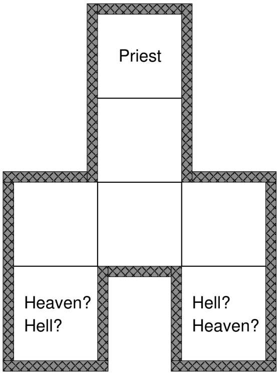


Figure 5.1: Multiagent Heaven or Hell environment. The priest (top state) can tell the agents where heaven is located (bottom-left or bottom-right state).

- The reward function $R(s, \bar{a})$ gives the team of agents a scalar reward at each time step for taking joint action a in state s. It is defined as the sum of each agent's local reward function and the joint reward of the team:

$$R(s, \bar{a}) = \sum_{i=1\ldots n} R_i(\bar{s}_i, a_i) + R(s_0, \ldots, s_m, a_1, \ldots, a_m). \tag{5.2}$$

The local reward $R_i(\bar{s}_i, a_i)$ of agent i is defined as the sum of the reward for the domain-level action $R_i(\bar{s}_i, a_i^d)$ and the communication cost $R_i(\bar{s}_i, a_i^\sigma)$:

$$R_i(\bar{s}_i, a_i^\sigma) = \begin{cases} 0 & \text{if } \sigma_i = \emptyset, \\ r_c < 0 & \text{otherwise.} \end{cases} \tag{5.3}$$

- Initial belief b_i^0, discount rate γ, and planning horizon h as in Sec. 2.4.2.

In the remainder of this chapter we shall limit our discussion to two agents, but our methods and experiments generalize to more than two agents. Broadcast communication is used, and we will focus on noise-free communication, fixing $p(o_i^\sigma|\bar{\sigma})$ to be the identity matrix.

To give some intuition on the range of problems our model captures we will now discuss a very simple example application. The "Multiagent Heaven or Hell" problem (inspired by Thrun, 2000) is a scenario in which two agents have to meet in one of two locations, one called "heaven" and the other one "hell", see Fig. 5.1. At the start of each trial heaven is positioned either in the bottom-left state ($s_0 = \textit{left}$) or in the bottom-right state ($s_0 = \textit{right}$) with equal probability, and hell will be located in the other state. If the agents meet in heaven the team receives a reward of $r_h > 0$, but meeting in hell is penalized with a reward of

$-r_h$. However, each agent does not know the location of heaven or hell until it visits the "priest" state, in which it receives an observation indicating whether heaven is left or right. Visiting the priest results in an individual negative reward of $r_p < 0$ for the particular agent, and we assume that the reward obtainable in heaven makes visiting the priest a viable option, i.e., $r_h > -mr_p$, where $m = 2$.

If no communication is possible, or the communication cost $(-r_c)$ is prohibitively large, the optimal policy could be for each agent to visit the priest, find out where heaven is located and go there. However, when communication is relatively cheap, i.e., $r_c > r_p$, more interesting scenarios become feasible. For instance, only agent i could ask the priest and inform agent j by sending it a message. This would save agent j the cost of visiting the priest and still be able to tell heaven from hell. However, instead of defining such semantics a priori, we would like the agents to learn when and how to communicate, and how to interpret incoming messages. In particular, the agents should optimize their use of communication with respect to the cost of sending a message r_c.

5.2.3 Belief tracking

In a general DEC-POMDP setting it is not possible for an agent to maintain an exact belief over the true state of the system, as it only receives its own part of the observation vector. As such, agent i has to reason about what observation agent j might have observed, but at the same time agent j is modeling agent i, which leads to the very high complexity class of solving DEC-POMDPs. As a result of the lack of independence between agents, only approximate beliefs can be computed (Gmytrasiewicz and Doshi, 2005). Alternatively, one can assume that agent i's observations are independent from those of agent j (Nair et al., 2003), which together with a common transition independence assumption precludes any use of direct communication in the DEC-POMDP model, as the agents cannot influence each other. In contrast, in our decentralized model the domain-level component o_i^d of an observation $o_i = (o_i^d, o_i^\sigma)$ only depends on an agent's local state $\bar{s}_i$, but the communication component o_i^σ does depend on agent j, as follows

$$p(o_i|\bar{s}_i, a_i) = p(o_i^d, o_i^\sigma|\bar{s}_i, a_i) \tag{5.4}$$

$$= p(o_i^d|\bar{s}_i, a_i)p(o_i^\sigma|\pi_j), \tag{5.5}$$

where $p(o_i^\sigma|\pi_j)$ is the probability that agent j sends agent i a particular message vector in the previous time step, which we will approximate based on its policy π_j. Consequently, we can only perform an approximate belief update, but it allows us to model a built-in communication channel.

Analogous to the POMDP belief update (2.11), repeated here for convenience:

$$b^{ao}(s') = \frac{p(o|s', a)}{p(o|b, a)} \sum_{s \in S} p(s'|s, a)b(s), \tag{2.11}$$

agent i's belief b_i is updated as follows when it takes action a_i and receives observation o_i:

$$b_i^{a_i o_i}(\bar{s}_i') = \frac{p(o_i^d|\bar{s}_i', a_i)p(o_i^\sigma|\pi_j)}{p(o_i|b_i, a_i)} \sum_{\bar{s}_i \in (S_0 \times S_i)} p(\bar{s}_i'|\bar{s}_i, a_i)b_i(\bar{s}_i), \tag{5.6}$$

where $p(o_i|b_i, a_i)$ is a normalizing constant. The probability that $b_i^{a_i o_i}$ will be the successor of b_i when agent i executes action a_i is defined as

$$\begin{aligned} p(b_i^{a_i o_i}|b_i, a_i) &= p(o_i|b_i, a_i) && (5.7)\\ &= \sum_{\bar{s}_i \in (S_0 \times S_i)} p(o_i|\bar{s}_i, a_i)b_i(\bar{s}_i). && (5.8) \end{aligned}$$

Since individual beliefs can be defined through (5.6), local policies π_i can be defined as in a standard POMDP setting, i.e., as mappings $\pi_i : \Delta(S_0 \times S_i) \to A_i$ from local beliefs b_i to actions. The only term not yet defined in (5.6) is $p(o_i^\sigma|\pi_j)$: the probability that agent i receives a message σ_j and interprets it as observation o_i^σ. It is defined by the probability that agent i receives σ_j as o_i^σ, as defined by the channel noise model $p(o_i^\sigma|\sigma_j)$, times the probability that agent j actually sends σ_j. The latter probability is estimated by computing the set of possible beliefs $\{b_{j,t}^\sigma\}$ in which agent j sends σ_j according to π_j, and summing the probabilities $p(b_{j,t})$ that any of those beliefs in $b_{j,t} \in \{b_{j,t}^\sigma\}$ will occur:

$$\begin{aligned} p_{t+1}(o_i^\sigma|\pi_j) &= p(o_i^\sigma|\sigma_j) \sum_{b_{j,t} \in \{b_{j,t}^\sigma\}} p(b_{j,t}), \ \text{ with} && (5.9)\\ \{b_{j,t}^\sigma\} &= \{b_{j,t}|(a_j^d, a_j^\sigma) = \pi_j(b_{j,t}), a_j^\sigma = \sigma_j\}. && (5.10) \end{aligned}$$

where $p(b_{j,t})$ is the probability that agent j is in belief $b_{j,t}$ at time t. The following equation computes $p(b_{j,t})$ as a recursion from time step 0, at which we know agent j is in its initial belief b_j^0 with probability 1:

$$p(b_{j,t}) = \begin{cases} 1 & \text{if } t = 0 \\ p(b_{j,t}|b_{j,t-1}, \pi_j(b_{j,t-1}))p(b_{j,t-1}) & \text{otherwise} \end{cases} \tag{5.11}$$

which uses (5.8). The set $\{b_{j,t}\}$ of all possible beliefs which agent j could believe to be true at a particular time step is finite for any given t, as we have assumed finite models.

5.2.4 Communication

The previous section defined how each agent updates its belief and how agent j's policy affects agent i's belief update, and here we provide the intuition how this implements a communication channel. For a particular incoming message σ_j agent i can use its knowledge of π_j and the set of beliefs $\{b_{j,t}\}$ for agent j to compute the set $\{b^{\sigma}_{j,t}\}$ (5.10) from which agent j would send the particular message σ_j. In this way, receiving message σ_j at time $t+1$ informs agent i about agent j's local state at time t:

$$p^{\sigma_j}_{t+1}(\bar{s}_j) = \sum_{b_{j,t} \in \{b^{\sigma}_{j,t}\}} b_{j,t}(\bar{s}_j) p(b_{j,t}), \tag{5.12}$$

combining (5.11) and (5.10). The crucial point is that, as agent j's local state $\bar{s}_j$ includes s_0, receiving message σ_j will provide agent i with information about s_0:

$$p^{\sigma_j}_{t+1}(s_0) = \sum_{s_j \in S_j} p^{\sigma_j}_{t+1}(\bar{s}_j) \ \text{ with } \ \bar{s}_j = (s_0, s_j). \tag{5.13}$$

In summary, the fact that s_0 is shared by all agents allows for communication-based learning, by enabling an agent j to modify the observation model of some other agent i, via the term $p(o^{\sigma}_i|\pi_j)$ in (5.6). The above analysis shows that the latter term is conditional on s_0, and therefore the term $p(o^{\sigma}_i|\pi_j)$ can non-trivially modify the belief of agent i.

Depending on the quality of agent j's communication policy, communication can improve team performance. For instance, if agent j communicates uniformly at random, agent i will not benefit from communication as the information content of the message is zero. However, in the Multiagent Heaven or Hell example described above, if agent j communicates $\sigma_j = 1$ if it knows heaven is left, communicates $\sigma_j = 2$ if it knows heaven is right, and does not communicate otherwise, then agent i can use σ_j to update its belief about the location of heaven in a useful manner and proceed to heaven without visiting the priest.

5.2.5 Solution method

After proposing a decentralized model for planning for a team of communicating agents, we will now present an approach for computing a plan in our model. We propose to compute a joint policy for the team of agents in an iterative fashion (Nair et al., 2003; Emery-Montemerlo et al., 2004; Bernstein et al., 2005). We start with a (randomly selected) policy π_i for each agent i. We fix the policy for all agents except for one, our protagonist agent i. From agent i's perspective we can treat the other agents as part of the environment, and agent i can build a POMDP model of this environment, given that it knows the policies of the other agents. For the resulting POMDP model it computes a policy using a standard (approximate) POMDP solver which replaces its current policy π_i. We then move on to the next agent and repeat the procedure from its perspective. This process is iterated until the performance of the joint policy does not improve anymore. In our case, this might not be a local maximum, as updating the communication policy of one agent can change the belief update of other agents. Multiple random restarts are necessary in general to reach good performance. We will solve the POMDP models using Perseus (see Chapter 3). It computes a policy in the form of a value function $V_i : \Delta(S_0 \times S_i) \rightarrow \mathbb{R}$, which for every belief estimates the amount of future discounted reward the agent can obtain.

5.2.5.1 Computing an agent's POMDP model

A crucial component is building the POMDP model from the known problem description and the policies of all teammates. In this POMDP model the reward function for agent i should not only consist of its local reward function R_i but also the expected contribution of the joint reward function R. For instance, in the Multiagent Heaven or Hell example described in Sec. 5.2.2, if agent i predicts that agent j's policy directs j to the heaven location, agent i's local reward function should report that if i goes to heaven too, it can receive the joint reward of r_h. We summarize the influence of the other agent by computing two statistics: $p(a_j|\bar{s}_j)$ is the probability agent j will execute a_j in $\bar{s}_j$, and $p(\bar{s}_j)$ is the probability that agent j will be in a particular state $\bar{s}_j$. We approximate these probabilities by simulating the belief tree for the team, given the pair of policies $\{\pi_i, \pi_j\}$. The root of the belief tree at $t = 0$ is the starting belief b_i^0 of each agent i. We compute the set of possible $t + 1$ beliefs for each agent by propagating each of its beliefs at time t:

$$\{b_{i,t+1}\} = \{b_{i,t}^{a_i o_i} | a_i = \pi_i(b_{i,t}), o_i \in O_i\}. \tag{5.14}$$

In order to keep the memory requirements at acceptable levels, some pruning of the belief tree will be necessary in general. From the belief sets at time step $\{b_{i,t}\}$ and the probability that each of the beliefs will be encountered (according to (5.11)) we can approximate the required statistics $p(a_j|\bar{s}_j)$ and $p(\bar{s}_j)$.

We use these statistics to define the local reward function R_{i,π_j} for agent i, which takes into account the expected contribution from agent j's policy π_j:

$$R_{i,\pi_j}(\bar{s}_i, a_i) = R_i(\bar{s}_i, a_i) + \sum_{\bar{s}_j \in (S_0 \times S_j), a_j \in A_j} R(s_0, s_i, s_j, a_i, a_j) p(a_j|\bar{s}_j) p(\bar{s}_j), \tag{5.15}$$

with $(s_0, s_i) = \bar{s}_i$. From the same statistics the observation model for agent i can be computed, which will encode the information contained in incoming messages from agent j, as we described in Sec. 5.2.4. To complete the POMDP model, the transition model for agent i is given by

$$p(\bar{s}_i'|\bar{s}_i, a_i) = p(s_0', s_i'|s_0, s_i, a_i^d, a_i^\sigma) \tag{5.16}$$

$$= p(s_0'|s_0) p(s_i'|s_i, a_i^d). \tag{5.17}$$

5.2.5.2 Learning to communicate

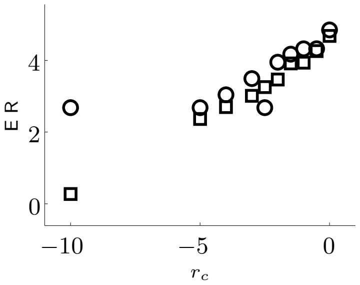
The iterative scheme described above is able to react to any incoming communication, which are incorporated in the local reward model and exploited by the POMDP solver. However, it will not learn from scratch to send any messages, as the other agent's policy has not yet learned how it can benefit from incoming messages. Put otherwise, if agent j's policy has not yet learned to respond to agent i's messages, then the expected contribution of π_j to agent i's reward model (5.15) will be zero. After solving agent i's POMDP, π_i will no longer send any messages, as each sending a message has a certain cost and the expected gain is not yet represented in agent i's reward model.

Therefore, we begin by equipping each agent with a fixed heuristic communication policy and iteratively optimize the agents' domain-level component of their policies to exploit the incoming messages. Next we aim to optimize the communication component of each agent's policy with respect to the communication reward $r_c < 0$. In particular, initially all agents could at each time step send the last domain-level observation they received. Choosing the domain-level observations as initial language is an appropriate choice, as it is known to be an optimal language for DEC-MDPs with constant message cost (Goldman and Zilberstein, 2004). However, we could improve team performance by only sending messages that are useful.

A message is useful when it conveys some information which the other agent can use to improve team performance. As the meaning of a message is defined over s_0, the entropy of $p^{\sigma_j}(s_0)$ (5.13) measures the information content of a particular message. However, if a message has low entropy does not necessarily mean it is useful to send. One proposal for testing its value is to modify each agent's local reward model (5.15) to prefer communicating low entropy messages to not sending them, but only at states in which it would send the messages under the initial heuristic communication policy. This ensures the other agent will know how to respond to the incoming message. Next we repeat the process of iterative optimization, allowing the agents to adjust to the new communication behavior.

5.3 Experiments

We will present experimental results in the Multiagent Heaven or Hell domain and in a game of tag, in which agents have to cooperate in order to tag an opponent. In these experiments we included the time step of the system in s_0, which allows for a single message to have a different meaning, depending on which time step it was sent. Adding time to s_0 requires no extra assumptions, but does allow the agents to more closely correlate their behavior. Providing correlation for agents in a DEC-POMDP has been shown to improve team performance (Bernstein et al., 2005).



(a) Reward.

r_c	2 agents	3 agents
0.0	14.00	21.00
−0.5	0.50	0.49
−1.0	1.00	0.49
−1.5	0.50	0.49
−2.0	0.50	0.49
−2.5	0.50	0.49
−3.0	0.50	0.51
−4.0	0.00	0.49
−5.0	0.00	0.49
−10.0	0.00	0.00

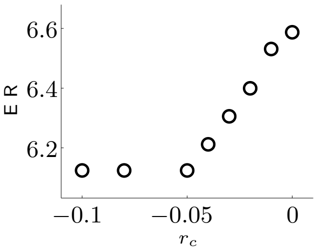
(b) No. of messages.

Figure 5.2: Multiagent Heaven or Hell results. The circles indicate results for 2 agents, and the squares show results for a team of 3 agents. Plot (a) shows the expected discounted reward (E R) obtained for different communication rewards (r_c), and table (b) lists the average number of messages sent.

5.3.1 Multiagent Heaven or Hell

First we will consider the Multiagent Heaven or Hell domain as introduced in Sec. 5.2.2. The environment contains seven grid cells ($|S_i| = 7$) and the shared state s_0 is the cross-product of the time index and {"heaven is in bottom-left grid cell","heaven is in bottom-right grid cell"}, where we set the horizon of the problem to 15. Each agent has a domain-level action set A_i^d = {*north, east, south, west, stay in place*}, where the first four are movement actions that transport the agent one grid cell in the corresponding direction, and the last action is a no-op which has no effect on the agent's location. Each agent can observe whether there is a wall to the left or to the right of it, and in the priest state it can observe the location of heaven, resulting in six possible observations O_i^d ={*left, right, both, neither, heaven-left, heaven-right*}. Meeting in heaven by all agents is rewarded by $r_h = 10$ and meeting in hell by $-r_h$. Visiting the priest incurs a negative reward $r_p = -2$, every movement action a reward of −0.1, and the discount factor γ is set to 0.95. Both the transition and observation model are deterministic, and the agents always start in the center grid cell (the cell at the T-intersection).

Fig. 5.2 shows results obtained by using the local reward modification scheme described in Sec. 5.2.5.2, for teams of 2 and 3 agents. First, we see in Fig. 5.2(a) that the team performance increases when the penalty for communication decreases, i.e., r_c becomes less negative. In all cases one agent i will visit the priest, but it will only send a message indicating the location of heaven if the communication cost is low ($r_c \geq -2$), see Fig. 5.2(b). Otherwise the other agent(s) will go to one of the heaven/hell locations, and agent i will only join them if it is



(a) Reward.

r_c	2 agents
0.00	10.67
−0.01	6.38
−0.02	10.67
−0.03	10.67
−0.04	10.67
−0.05	0.00
−0.08	0.00
−0.10	0.00

(b) No. of messages.

Figure 5.3: Pack Tag results. Plot (a) shows the expected discounted reward (E R) obtained for different communication rewards (r_c), and table (b) lists the average number of messages sent.

indeed the correct location. When communication is free, performance is best and a large number of messages are communicated, as one would expect. On the other extreme, when communication is expensive, the performance remains stable for the 2-agent case, but for 3 agents the algorithm gets trapped in a local maximum.

5.3.2 Pack Tag

The "Pack Tag" problem models a game of tag, in which two robots have to tag an opponent robot (Pineau et al., 2003a; Emery-Montemerlo et al., 2004). The game is played on a $k \times k$ grid, with one robot starting in the bottom-left corner and the other one in the top-right corner. At the beginning of each game the opponent is positioned uniformly at random in one of the grid cells and remains stationary. The two robots have to occupy the same grid cell as the opponent and execute the *tag* action simultaneously in order to tag the opponent. The robots can move in any of the four compass directions, and their actions are deterministic. The two robots have a sensor which detects the opponent with full certainty when the opponent is sharing the same grid cell. Each robot receives a null observation otherwise, but remains perfectly localized due to the known starting position and noise-free motion. Each action incurs a reward of -0.1 and successfully tagging the opponent results in a reward of 10. The agents have to tag the opponent in 10 time steps, and γ was set to 0.95.

As in Sec. 5.3.1 we varied the communication cost while employing the local reward modification method, and using a grid size $k = 3$. Fig. 5.3 shows that when the communication cost is high, $r_c \leq -0.05$, the agents converge to a non-communicating policy pair. They both search the grid, and when one of them sees the opponent it waits for the other to join it before jointly tagging it. However, when communication is cheaper the agents communicate when they observe the opponents, which indicates to the other agent the location of the opponent. Here the benefit of including time in s_0 is manifested, as receiving the same message at a different time step has a different meaning: agent i knows agent j's policy, and therefore knows j's search pattern, and in this way knows in what grid cell agent j observed the opponent when j send its message. After receiving an informative message an agent abandons its search pattern and moves directly to its teammate's position, where they jointly tag the opponent. The policy pair computed for $r_c = -0.01$ has converged to a better local maximum, which uses a more efficient form of communication.

5.4 Discussion

In this chapter we considered the problem of planning for cooperative multiagent systems. If the task assigned to the system requires tight coordination at all times, it is natural to treat the team as a single agent, and plan in the space of joint actions. Instead, we focused on multiagent problems in which explicit cooperation is required at some stages, but at other points in time and space the agents can act in a more independent fashion. We studied the problem of computing a plan for a team of agents inhabiting a stochastic environment that is only partially observable to them. The agents have the capability to communicate, but sending messages has a certain cost and the available bandwidth is limited. Decentralized partially observable Markov decision processes (DEC-POMDPs) formalize such multiagent planning problems for groups of cooperating agents. We presented a decentralized model for tackling these kinds of planning problems, which incorporates a communication channel.

When comparing with other recent approaches to planning for cooperative multiagent systems in partially observable environments, there are two main advantages to the proposed model. First of all, we treat communication as an integral part of an agent's reasoning, not as an add-on. In particular, we do not define a priori semantics for a message, but treat it as part of the optimization problem. The meaning of a particular message σ_i is defined by the situations in which agent i will send σ_i. Of course, assuming a message set Σ_i with user-defined, high quality and fixed semantics simplifies the optimization problem, but we believe it is worthwhile to explore optimizing communication as part of (approximately) solving the DEC-POMDP.

The second feature of the proposed model is that it is decentralized, and does not require an (approximate) solution to the large single-agent POMDP resulting from truly free communication (as described in Sec. 5.2.1). The centralized state, action and observation spaces grow exponentially in the number of agents, rendering even approximate solutions infeasible. In our decentralized model, only the observation set for each agent grows with the number of agents, which can be limited by manipulating the size of possible messages Σ_i. One other possible solution to combat this exponential growth would be to assume that messages from different agents are conditionally independent given the state, which would allow us to factorize each agent's observation model. A factorized observation model induces a particular structure in an agent's POMDP, which can facilitate solving the POMDP.

We trade off exploiting an (approximately) optimal centralized solution for a potentially more scalable decentralized setting, in which each agent only considers its own local state (and some shared uncontrollable state features). Furthermore, a decentralized framework is more natural for modeling agents who cannot observe each other's state nor have truly free communication available. We proposed an approximate iterative method for computing policies in the proposed model, and obtained encouraging preliminary experimental results in two domains.

As future work we would like to examine possibilities of simultaneous learning of communication-derived POMDPs for two or more agents, which could potentially improve upon the alternating-maximization paradigm. Furthermore, we would like to experiment on problems with more agents, which are too large to be solved by centralized methods. For problems that are still solvable by centralized algorithms, we intend to compare the centralized solution to the decentralized one, to gain more insight in how optimality is traded off for more efficient computation.

Finally, an interesting avenue of future work would be to study how scalable multiagent coordination techniques such as coordination graphs (Guestrin, Koller, and Parr, 2002a; Kok, Spaan, and Vlassis, 2003; Kok et al., 2005) can be transferred to a DEC-POMDP setting. In this context, Nair, Varakantham, Tambe, and Yokoo (2005) have proposed Networked Distributed POMDPs, which combines DEC-POMDP methods with distributed constraint optimization techniques (Modi, Shen, Tambe, and Yokoo, 2003), in order to exploit local interactions between agents. Coordination graphs are based on the same premise, i.e., the assumption that global coordination problems often can be successfully approximated by only considering local coordination problems.

Chapter 6

Conclusions

Planning under uncertainty is an important topic in artificial intelligence, as it enables the design of agents that can perform a sequential decision-making task as well as possible, even in uncertain environments. In this final chapter we will present concluding remarks on the work described in this thesis, and outline its contributions. In Sec. 6.2 we will discuss numerous application domains in which our techniques could be or have been applied, ranging from industrial settings such as machine maintenance to robotic, marketing, and medical applications. Finally, Sec. 6.3 will outline several promising avenues of future research.

6.1 Conclusions and contributions

In this thesis we have studied several instances of the problem of planning under uncertainty, which forms a key technology for the design of intelligent agents. An intelligent agent, whether robotic, human, or simulated, should be able to autonomously perform a given task. An agent acts according to a plan, which maps sensory input to the optimal action to execute for the given task. We have considered computing plans for single agents as well as cooperative multiagent systems, in domains in which an agent is uncertain about the exact consequence of its actions. Furthermore, we assumed that the agent is equipped with imperfect sensors, resulting in noisy sensor readings that provide only limited information. For single agents, such planning problems are naturally framed in the partially observable Markov decision process (POMDP) paradigm.

We presented Perseus, a randomized point-based value-iteration algorithm for planning in POMDPs. Perseus operates on a large belief set sampled by simulating random trajectories through belief space. Approximate value iteration is performed on this belief set by applying a number of backup stages, ensuring that in each backup stage the value of each point in the belief set is improved; the key observation is that a single backup may improve the value of many belief points. Contrary to other point-based methods, PERSEUS backs up only a (randomly selected) subset of points in the belief set, sufficient for improving the value of each belief point in the set. Experiments confirm that this allows us to compute value functions that consist of only a small number of vectors (relative to the size of the belief set), leading to significant speedups. We performed experiments in benchmark problems from literature, and PERSEUS turns out to be very competitive to other methods in terms of solution quality and computation time. Additionally, we reported how PERSEUS can be applied to robotic planning problems. Robots typically have to deal with large state spaces, high-dimensional sensor readings, perceptual aliasing and uncertain actions. We defined a mail delivery task in which a simulated robot has to deliver mail in an office environment, and we proposed to compress and cluster the omnidirectional camera images the robot observes. PERSEUS can succesfully solve the resulting POMDP model, and the robot will pick up and deliver the mail from any starting location in the highly perceptually aliased office environment.

As PERSEUS requires no particular assumptions on the POMDP domain, it is widely applicable for a range of problems involving planning under uncertainty. Aside from the experimental domains we presented throughout this thesis, PERSEUS has been applied in several other planning problems. Boger et al. (2005) consider the real-world problem of assisting persons with dementia in activities of daily living. In particular, they model washing one's hands as a POMDP, and use PERSEUS to compute a policy that guides a user's handwashing activity. Williams and Young (2005); Williams, Poupart, and Young (2005a) model the interaction of a user with a computer system equipped with a speech-recognition system as a POMDP. They apply PERSEUS to compute dialogue-management policies in a travel domain, in which the user attempts to buy a ticket to travel from one city to another one. The system asks the user a series of questions, followed by a ticket purchase request, which ends the dialogue. Poupart (2005) uses PERSEUS to compute a policy for managing a network of computers. The goal is to keep as much computers running as possible, and the system's actions are doing nothing, pinging a machine or rebooting a machine. Some of these applications tackle very large POMDPs with 50 million states, which require alternative representations for the POMDP model and solution.

Extending PERSEUS to alternative representations is possible as the main idea of PERSEUS—improving the value of set of sampled beliefs by randomly updating belief points—is relatively independent of the particular POMDP and value-function representation. Poupart (2005) extends PERSEUS to compute policies represented compactly as algebraic decisions diagrams instead of vectors. A complementary way to scale up PERSEUS to such very large domains is to apply linear compression techniques such as value-directed compression

(Poupart and Boutilier, 2003b) on the POMDP model, which allows for direct application of approximate POMDP techniques such as Perseus (Poupart, 2005). Perseus has been used in model-free problem settings, to quickly recompute policies as more experience regarding the model is gained (Shani et al., 2005a; Shani, Brafman, and Shimony, 2005b), and in a Bayesian reinforcement-learning setting (Poupart et al., 2006). Perseus has also been extended to use predictive state representations as planning framework (James et al., 2006), and to work with continuous observation spaces (Hoey and Poupart, 2005).

In Chapter 4, we extended Perseus to compute plans for agents which have a very large or continuous set of actions at their disposal, by sampling actions from the action space. When sampling a belief in the Perseus backup stage, we also sample a number of actions, compute the next-horizon vectors for the particular belief, and we keep the vector which improves the value of the belief most. We demonstrated the viability of Perseus on two POMDP problems with continuous action spaces: a continuous navigation task and a robotic problem involving a mobile robot with omnidirectional vision. We analyzed a number of different action sampling schemes and compared with discretized action spaces. We demonstrated that Perseus can compute successful policies for these domains while sampling from a continuous set of actions, and can outperform Perseus with coarse discrete action spaces. We investigated the effect of various sampling schemes, and we showed that exploring actions are selected in the early backup stages, but that over time actions that turn out to be useful are exploited.

Many real-world POMDPs are naturally defined using continuous states, for instance the location of a robot, but the large majority of POMDP solution techniques assume discrete state spaces. We demonstrated the piecewise linearity and convexity of value functions defined over infinite-dimensional belief states induced by continuous states. As it can be shown that Bellman backups in continuous state spaces are isotonic and contracting, value iteration can be extended to continuous-state POMDPs. However, we need a representation that allows analytical computation of integrals and that is also closed under Bellman backups. Linear combinations of Gaussians form a suitable and expressive representation, and we extended Perseus to operate on Gaussian-based POMDP models. We demonstrated our method on a simple robot navigation task, in which a simulated robot succesfully navigates a perceptually aliased corridor.

In Chapter 5 we studied the problem of computing a plan for a team of agents. We considered a general problem setting in which agents inhabit a stochastic environment that is only partially observable to them. The agents have the capability to communicate, but sending messages has a certain cost and the available bandwidth is limited. Decentralized partially observable Markov decision processes (DEC-POMDPs) formalize such multiagent planning problems for groups of cooperating agents. We presented a decentralized model for tackling these kinds of planning problems, which incorporates a communication channel. Decentralized models potentially scale better and are a more natural paradigm for modeling agents that cannot observe each other's state. Contrary to other recent approaches, we treat communication as an integral part of the model, and model it directly in the DEC-POMDP. We proposed an approximate iterative method for computing policies in the proposed model. We obtained encouraging preliminary experimental results in two domains, a simple toy domain and in a game of tag, in which agents have to cooperate in order to tag an opponent.

6.2 Applications

In this thesis we did not focus on a particular application domain, but instead we developed algorithms that can be applied to many different planning domains. In this section we will not try to exhaustively list every potential application domain. Instead, we will provide a broad overview of problems that have been (or could be) tackled by decision-theoretic planning methods, such as the ones presented in this thesis. Cassandra (1998b) provides an early thorough overview of POMDP applications.

Initial POMDP applications stem from operations research and have an industrial flavor, for instance the machine maintenance problem, which is the subject of early work on POMDPs (Ross, 1971; Smallwood and Sondik, 1973; White, 1977; David, Friedman, and Sinuany-Stern, 1999; Hsu and Arapostathis, 2004; Simmons Ivy and Black Nembhard, 2005). A machine is assumed to consist of a number of parts, which deteriorate over time, and each one can cause the machine to fail. The state of each part, i.e., how much it has been worn down, can only be inspected when the machine is stopped and dismantled. In that case, money is lost as the machine does not produce any goods. On the other hand, when a part fails the produced goods are lost. We can model such an optimal inspection/replacement problem as a POMDP, which trades off the cost of inspection, replacing parts, and the expected benefit of just continuing to operate the machine. Related to machine maintenance is the problem of structural inspection, e.g., when to inspect or perform maintenance on roads or bridges (Ellis, Jiang, and Corotis, 1995). Other operations-research type of applications are inventory control (Treharne and Sox, 2002) or computing dynamic pricing strategies (Aviv and Pazgal, 2005).

A range of POMDP applications consider modeling humans in their state description. Hauskrecht and Fraser (2000) describe a medical application for the management of patients, characterized by hidden disease states, and where the actions consist of diagnostic and treatment procedures. Hu, Lovejoy, and Shafer (1996) propose a POMDP model to investigate drug therapy policies for maintaining a patient's drug dosage at a certain level. Boger et al. (2005) apply Perseus in a real-world task for assisting persons with dementia, in which users receive verbal assistance while washing their hands. Rusmevichientong and Van Roy (2001) propose a POMDP model for optimizing marketing campaigns, which models a customer's responses to different products presented during an advertising campaign. The goal is to figure out a customer's profile, in order to be able to tailor the marketing campaign to the customer.

Spoken dialogue management considers the problem of humans interacting with computers or robots in form of a dialogue. The user's intention will often not be immediately clear to a robot due to noise in speech recognition or due to lack in the robot's knowledge of the user's context. Roy, Pineau, and Thrun (2000) consider a POMDP model for the dialogue management of a robotic assistant, and Williams and Young (2005); Williams et al. (2005a) apply Perseus for solving such dialogue-management POMDPs.

In Chapters 3 and 4 we have seen examples of how POMDPs can be used for robot localization and navigation, as has been shown before (Simmons and Koenig, 1995; Theocharous and Mahadevan, 2002; Theocharous, Murphy, and Kaelbling, 2004). POMDP models have been applied to high-level control of a robotic assistant, designed to interact with elderly people (Pineau et al., 2003b; Roy, Gordon, and Thrun, 2003). Darrell and Pentland (1996) propose a visual gesture recognition system, in which a POMDP controller steers the focus of the camera to regions in the image which are most likely to improve recognition performance.

Potential applications for decision-theoretic planning in a cooperative multiagent setting include cooperative robotics (Arai, Pagello, and Parker, 2002; Howard, Parker, and Sukhatme, 2004; Kok et al., 2005; Emery-Montemerlo, Gordon, Schneider, and Thrun, 2005), sensor networks (Lesser, Ortiz, and Tambe, 2003), distributed space applications (Dias, Stentz, and Goldberg, 2003), communication networks (Ooi and Wornell, 1996; Tao, Baxter, and Weaver, 2001; Altman, 2002) and supply chain management (Swaminathan, Smith, and Sadeh, 1998; Fox, Barbuceanu, and Teigen, 2000).

6.3 Future work

In this section, we will identify several different avenues of future research, expanding the work presented in this thesis. We will highlight potential extensions to Perseus, in particular alternative problem representations and a unified approximate POMDP planner for continuous domains. For the multiagent case, we will discuss several interesting possibilities for future work, including the combination of DEC-POMDP planning with coordination graphs.

Point-based POMDP techniques such as Perseus attempt to exploit structure present in planning domains by focusing computational resources on the reachable belief space. In the literature orthogonal techniques for exploiting problem structure exist, which are based on alternative problem representations. For instance, in many problems the state description can be factorized in a number of state features, as we discussed in Sec. 2.2.4. Instead of defining a transition function over the full state space, whose size is exponential in the number of state features, we can compactly represent the transition model using a dynamic Bayesian network. By also employing Bayesian network representations for the observation and reward models, we can define factored POMDPs for planning in large, structured problems (Boutilier and Poole, 1996; Hansen and Feng, 2000; Guestrin, Koller, and Parr, 2001; Poupart, 2005). Given such a factored POMDP representation, we will need to choose an appropriate value-function representation. Poupart (2005) extends Perseus to compute policies represented as algebraic decisions diagrams in factored POMDPs, and Guestrin et al. (2001) propose to represent value functions as linear combinations of basis functions, where each basis function only depends on a subset of all state features. This allows for approximate but compact representations, suitable for integration in Perseus.

Predictive state representations (PSRs) have been proposed as an alternative to POMDPs for modeling stochastic and partially observable environments (Littman et al., 2002; Singh et al., 2004). A PSR dispenses with the hidden POMDP states, and only considers sequences of action and observations which are observed quantities. In a PSR, the state of the system is expressed in possible future event sequences, or “core tests”, of alternating actions and observations. The state of a PSR is defined as a vector of probabilities that each core test can actually be realized, given the current history. The advantages of the predictive state representation are most apparent when trying to perform reinforcement learning in partially observable stochastic environments, as the PSR only considers observable events instead of hidden states. Perseus can be easily extended to plan in PSRs (James et al., 2006), as the PSR value function has been proven to be piecewise linear and convex (James, Singh, and Littman, 2004). An interesting line of research would be to further explore the application of point-based POMDP methods in PSR settings.

A different way of tackling the scale and complexity of real-world planning problems is to introduce hierarchy, which has been applied in many AI-related tasks, for instance when learning a map for robot navigation (Thrun, 1998; Zivkovic, Bakker, and Kröse, 2005). The general idea is to divide the problem into smaller subproblems, which are defined on a particular level of abstraction. Each subproblem is easier to solve than the original problem, and hierarchical approaches combine these partial solutions in a tree, where each lower-level solution provides the input and output for a higher-level solution. Also in the POMDP literature several hierarchical approaches have been proposed, both in a reinforcement-learning setting (Wiering and Schmidhuber, 1997; Hernandez-Gardiol and Mahadevan, 2001) and in a planning context (Pineau and Thrun, 2002; Theocharous et al., 2004; Foka and Trahanias, 2005). In the model-based setting we considered, hierarchical POMDPs have for instance been applied in robotic localization and navigation problems (Theocharous et al., 2004; Foka and Trahanias, 2005). As with compact problem representations, hierarchical approaches could prove an interesting direction of further research, in order to be able to scale approximate POMDP planning to larger real-world domains.

Many real-world POMDPs are naturally modeled by continuous states, actions and observations, as we discussed in Chapter 4. Given the numerous POMDP techniques for discrete models, a common approach for handling continuous models consists of discretizing or approximating the continuous components with a grid (Thrun, 2000; Roy et al., 2005). This usually leads to an important tradeoff between complexity and accuracy as we vary the coarseness of the discretization (or the grid). More precisely, as we refine a discretization (or grid), computational complexity increases. Alternatively, one can work directly with continuous components, as we proposed in Chapter 4. We investigated continuous-state POMDPs in which the models and beliefs are represented using linear combinations of Gaussian distributions, allowing for exact computation of Bellman backups. We also tackled the problem of planning with continuous actions by sampling the action space, and highlighted work by Hoey and Poupart (2005) on point-based POMDP planning with continuous observation spaces. As these three extensions are orthogonal in nature, a natural and promising avenue of future research would be to unify them, in order to create an approximate POMDP planner for fully continuous domains. Another interesting research direction would be to investigate which families of functions (beyond mixtures of Gaussians) are closed under Bellman backups and belief updates for different types of transition, observation and reward models. Also, we would like to consider an alternative representation for the belief densities, by representing a belief using a number of random samples, or particles, instead of a Gaussian mixture. Particle-based representations have been very popular in recent years, and they have been used in many applications ranging from tracking to simultaneous localization and mapping (see Doucet, de Freitas, and Gordon, 2001, for a review).

An interesting focus for future work is Bayesian reinforcement learning, which can be cast as a POMDP with a continuous state space (Sec. 4.4.2; Duff, 2002). It would be interesting to solve Bayesian reinforcement-learning problems using a similar approach as the Gaussian continuous-state approximate POMDP solver presented in Sec. 4.3. Poupart et al. (2006) propose such an extension to Perseus, in which the value function is composed of multivariate polynomials, a representation which they prove is closed under Bellman backups. This parameterization allows for offline policy optimization, while the online learning is computationally cheap as it only requires belief monitoring. Unfortunately, as all optimization is done offline, the amount of computation becomes prohibitive for large MDPs, requiring the user to limit the number of free parameters or provide informative priors. It would be interesting to study how such offline optimization techniques can be combined with online ones, for instance the sparse sampling method proposed by Wang, Lizotte, Bowling, and Schuurmans (2005).

When considering the model for planning under uncertainty for teams of agents we presented in Chapter 5, we can identify many interesting avenues of future work. In contrast with the large body of literature on POMDPs, accumulated over more than 30 years, DEC-POMDP models have only recently been receiving considerable attention, no doubt due to their high complexity. One promising line of research would be to consider alternatives for the alternating-maximization paradigm, in order to be able to simultaneously learn communication-derived POMDPs for two or more agents. Also, in the model we presented each agent's state and action space are local, but the observation space still grows exponentially in the number of agents, as an agent considers all possible combinations of incoming messages. An interesting refinement of the model would be to treat incoming messages from each agent as independent, given the state of those agents. Such an assumption is reasonable, as each message's meaning is defined over the state, and would induce a particular structure in the optimization problem. In particular, the POMDP from each agent's perspective would have several stages, in which the agent's domain level observation and all incoming messages are processed sequentially, thereby avoiding an exponentially sized observation space.

Finally, an interesting direction of future research would be to consider how scalable multiagent coordination techniques such as coordination graphs (Guestrin et al., 2002a; Kok et al., 2005) can be transferred to a DEC-POMDP setting. Coordination refers to the process that ensures that the individual decisions of the agents result in jointly optimal decisions for the group. The joint-action space grows exponentially with the number of agents, but the particular structure of the coordination problem can often be exploited to reduce its complexity. Our focus in Chapter 5 has been on domains in which at a given time step only a limited number of dependencies between agents exists. Via a context-specific decomposition of the problem into smaller subproblems, coordination graphs offer scalable solutions to the problem of multiagent decision making (Guestrin et al., 2002a; Guestrin, Venkataraman, and Koller, 2002b). We have successfully applied coordination graphs to continuous and dynamic domains such as the RoboCup soccer simulation by assigning roles to the agents and then coordinating the different roles (Kok et al., 2003, 2005). In this context, Nair et al. (2005) have proposed Networked Distributed POMDPs, which combines DEC-POMDP methods with distributed constraint-optimization techniques (Modi et al., 2003), in order to exploit local interactions between agents. Coordination graphs are based on the same premise, i.e., the assumption that global coordination problems often can be successfully approximated by only considering local coordination problems. Combining coordination graphs and DEC-POMDPs might prove fruitful for scaling up multiagent planning under uncertainty.

Appendix A

Gaussian mixture condensation

The components that represent beliefs and α functions are used in all basic operations of continuous-state PERSEUS, so for an efficient implementation of the algorithm we need to keep the number of components reasonably bounded. To achieve this objective, we use the procedure presented by Goldberger and Roweis (2005), which transforms a given Gaussian mixture f with k components to another Gaussian mixture g with at most m components, $m < k$, while retaining the initial component structure. The method directly manipulates the model parameters, and avoids the need to sample from the mixtures. The Gaussian mixture condensation algorithm is detailed in Algorithm A.1, where ϵ is a sufficiently small threshold.

The algorithm uses the Kullback-Leibler, $KL(f_i \| g_i)$, distance between two Gaussian distributions $f_i = N(\mu, \Sigma)$, $g_j = N(\mu', \Sigma')$, which is defined as

$$KL(f_i \| g_j) = \frac{1}{2} \left(\log \frac{|\Sigma'|}{|\Sigma|} + Tr((\Sigma')^{-1}\Sigma) + (\mu - \mu')^\top (\Sigma')^{-1} (\mu - \mu') - c \right), \tag{A.1}$$

with c the dimensionality of the space in which the Gaussian distributions are defined.

Observe that Algorithm A.1 is defined for normalized Gaussian mixtures and our α functions are unnormalized Gaussian mixtures. Therefore, for the α function compression, we use a modified version of the procedure in which the weights are normalized after taking their absolute value, to ensure that relevant value peaks, both negative or positive ones, are preserved. After the compression, the inverse procedure is applied to recover weights in the original scale.

Algorithm A.1 Gaussian mixture condensation(f,m), by Goldberger and Roweis (2005)

```
Input is a Gaussian mixture $f = \sum_{i=1}^{k} w_i\, f_i(x|\mu_i, \Sigma_i)$ and $m < k$.
Returns a Gaussian mixture $g = \sum_{i=1}^{m} w'_i\, g_i(x|\mu'_i, \Sigma'_i)$.
for all $j \in [1, m]$ do {Initialize}
  $w'_j \leftarrow w_j, \quad \mu'_j \leftarrow \mu_j, \quad \Sigma'_j \leftarrow \Sigma_j$
$d \leftarrow \sum_{i=1}^{k} w_i\ \min_{j \in [1,m]} KL(f_i \| g_j)$
repeat
  for all $i \in [1, k]$ do {Compute the mapping from $f$ to $g$}
    $\pi(i) \leftarrow \arg\min_{j \in [1,m], w'_j > 0} KL(f_i \| g_j)$
  for all $j \in [1, m]$ do {Define a new $g$}
    $I_j \leftarrow \{i \mid \pi(i) = j,\ i \in [1, k]\}$
    $w'_j \leftarrow \sum_{i \in I_j} w_i$
    $\mu'_j \leftarrow \frac{1}{w'_j} \sum_{i \in I_j} w_i\, \mu_i$
    $\Sigma'_j \leftarrow \frac{1}{w'_j} \sum_{i \in I_j} w_i\, (\Sigma_i + (\mu_i - \mu'_j)(\mu_i - \mu'_j)^\top)$
  $d' \leftarrow d$
  $d \leftarrow \sum_{i=1}^{k} w_i\, KL(f_i \| g_{\pi(i)})$
until $\frac{|d - d'|}{d} < \epsilon$
Return $g$
```

Summary

A major goal of artificial intelligence (AI) is to build agents: systems that perceive their environment and execute actions. For instance, a mobile robot might have a camera to observe its surroundings and wheels to move around. In particular, AI aims to develop intelligent agents, which attempt to perform an assigned task as well as possible. The agent acts according to a plan, which maps sensory input to the optimal action to execute for the task.

In this thesis, we have focused on computing plans for single agents as well as cooperative multiagent systems, in domains in which an agent is uncertain about the exact consequences of its actions. Furthermore, it is equipped with imperfect sensors, resulting in noisy sensor readings which provide only limited information. For single agents, such planning problems are naturally framed in the partially observable Markov decision process (POMDP) paradigm, which we have adopted throughout this thesis. In a POMDP, uncertainty in acting and sensing is captured in probabilistic models, and allows an agent to plan on its belief state, which summarizes all the information the agent has received regarding its environment.

As optimal planning in POMDPs is intractable in general we have focused on approximate but tractable methods. In Chapter 3 we considered a recent line of research on approximate point-based POMDP algorithms that plan on a sampled set of belief points. We presented Perseus, a randomized point-based value-iteration algorithm for planning in POMDPs, which exploits structure present in real-world planning domains. We performed experiments in benchmark problems from literature, and Perseus turns out to be very competitive to other methods in terms of solution quality and computation time. Additionally, we have shown successful results from an office delivery task involving a mobile robot with omnidirectional vision in a highly perceptually aliased office environment.

In Chapter 4, we extended Perseus to compute plans for agents which have a very large or continuous set of actions at their disposal, by sampling actions from the action space, and keeping those actions which turn out to be useful. We demonstrated the viability of Perseus on two POMDP problems with

continuous action spaces: a continuous navigation task and a robotic problem involving a mobile robot with omnidirectional vision. For POMDPs with continuous states, we demonstrated the piecewise linearity and convexity of value functions defined over infinite-dimensional belief states induced by continuous states. We extended Perseus to plan in continuous-state POMDP models represented by linear combinations of Gaussian distributions, which form a suitable and expressive representation that is closed under Bellman backups. We demonstrated our method by successfully applying it to a simple robot navigation task. We concluded the chapter by highlighting several third-party extensions to Perseus.

We switched to the multiagent setting in Chapter 5, in which we considered the problem of cooperative multiagent planning under uncertainty. We adopted the decentralized POMDP (DEC-POMDP) framework to compute plans for a team of agents that have the ability to communicate, but with limited bandwidth and at a certain cost. We proposed a decentralized model, in which each agent only reasons about its own local state and some uncontrollable state features, which are shared by all team members. In contrast to other approaches, we model communication as an integral part of the agent's reasoning, in which the meaning of a message is directly encoded in the policy of the communicating agent.

Finally, in Chapter 6 we presented general conclusions, and outlined promising directions of future research. We also highlighted potential application areas in which the methods presented in this thesis have been or could be applied.

Samenvatting[1]

Een van de hoofddoelen van onderzoek naar kunstmatige intelligentie is het ontwikkelen van "agenten": systemen die hun omgeving kunnen waarnemen en acties kunnen uitvoeren. Een mobiele robot kan bijvoorbeeld met een camera zijn omgeving observeren en zich voortbewegen door zijn wielen aan te sturen. Kunstmatige intelligentie richt zich met name op het ontwerpen van intelligente agenten, die een opgelegde taak zo goed mogelijk proberen uit te voeren. Een agent kiest zijn acties volgens een plan, dat voor iedere mogelijke sensorwaarneming voorschrijft wat de beste actie is, gelet op de taak van de agent.

In dit proefschrift zijn we geïnteresseerd in het plannen voor zowel enkele agenten alsmede voor teams van meerdere, samenwerkende agenten. We richten ons op agenten die onzeker zijn over de precieze gevolgen van het uitvoeren van acties. Ook beschikt de agent over sensoren met slechts een beperkt bereik en een begrensde nauwkeurigheid. Voor individuele agenten kan deze probleemstelling worden aangepakt binnen het raamwerk van "partially observable Markov decision processes" (POMDPs), dat de basis vormt van het onderzoek in dit proefschrift. In een POMDP worden de onzekerheden in sensorwaarnemingen en het uitvoeren van acties gerepresenteerd in kansmodellen, waardoor de agent kan plannen aan de hand van zijn geloof, of "belief", dat al zijn informatie over zijn omgeving samenvat.

Aangezien het exact berekenen van een optimaal plan in een POMDP zeer complex is, richten wij ons op handelbare algoritmes die een optimaal plan proberen te benaderen. In Hoofdstuk 3 hebben we een aantal zogenaamde punt-gebaseerde POMDP benaderingstechnieken behandeld, die een plan berekenen aan de hand van slechts een beperkte verzameling van mogelijke belief punten. We presenteerden PERSEUS, een punt-gebaseerd "value-iteration" algoritme om te plannen in POMDPs, dat structuur van realistische planningproblemen uitbuit. We hebben experimenten gedaan in standaard testproblemen uit de literatuur, en PERSEUS blijkt, in vergelijking met andere methoden, zeer goede prestaties te leveren gelet op kwaliteit van de gevonden oplossing en rekentijd. Verder hebben we succesvolle resultaten laten zien voor een gesimuleerde mobie-

[1] Summary in Dutch.

le robot met een omnidirectionele camera, die als taak heeft om post te bezorgen in een kantooromgeving.

In Hoofdstuk 4 hebben we Perseus uitgebreid voor het plannen voor agenten die een zeer grote of continue verzameling acties tot hun beschikking hebben door willekeurige acties uit deze actieruimte te proberen, en alleen acties te bewaren die nuttig blijken te zijn. We hebben de toepasbaarheid van Perseus gedemonstreerd in twee POMDP problemen met continue actieruimten: een continue navigatietaak en een planningprobleem voor een mobiele robot met omnidirectionele visie. Voor POMDPs met continue toestanden lieten we zien dat "value" functies opgebouwd zijn uit lineaire stukken en convex zijn over de oneindig-dimensionale belief toestanden. We hebben Perseus uitgebreid om te kunnen plannen in POMDPs met continue toestanden waarin alle modellen worden gerepresenteerd als lineaire combinaties van Gauss-verdelingen, hetgeen een adequate en expressieve representatie vormt die gesloten is onder "Bellman backups". We hebben onze aanpak gedemonstreerd op een eenvoudige robot navigatietaak. Het hoofdstuk werd afgesloten met een beschrijving van uitbreidingen op Perseus zoals die zijn ontwikkeld door derden.

In Hoofdstuk 5 behandelden we het probleem van plannen onder onzekerheid voor meerdere samenwerkende agenten. We gebruikten het gedecentraliseerde POMDP (DEC-POMDP) raamwerk om te plannen voor teams van agenten die met elkaar kunnen communiceren, maar waar de beschikbare bandbreedte beperkt is en communiceren bepaalde kosten met zich meebrengt. We presenteerden een gedecentraliseerd model, waarin iedere agent alleen maar rekening houdt met zijn eigen lokale toestand en enkele toestandskenmerken die door alle agenten gedeeld worden, maar die zij niet kunnen beïnvloeden. In tegenstelling tot andere methoden modeleren wij communicatie als een integraal onderdeel van het beslissingsproces van een agent, waarin de betekenis van een bericht wordt bepaald door het plan van de communicerende agent.

Tenslotte trokken we in Hoofdstuk 6 enige conclusies en belichtten we veelbelovende richtingen van mogelijk vervolgonderzoek. Verder bespraken we mogelijke toepassingsgebieden waar technieken uit dit proefschrift zouden kunnen worden gebruikt, of reeds toegepast zijn.

Bibliography

D. Aberdeen and J. Baxter. Scaling internal-state policy-gradient methods for POMDPs. In *International Conference on Machine Learning*, 2002.

E. Altman. Applications of Markov decision processes in communication networks. In E. A. Feinberg and A. Shwartz, editors, *Handbook of Markov Decision Processes: Methods and Applications*. Kluwer Academic Publishers, 2002.

C. Amato, D. S. Bernstein, and S. Zilberstein. Optimal fixed-size controllers for decentralized pomdps. In *AAMAS06 Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, 2006.

M. Aoki. Optimal control of partially observable Markovian systems. *Journal of The Franklin Institute*, 280(5):367–386, 1965.

T. Arai, E. Pagello, and L. E. Parker. Editorial: Advances in multi-robot systems. *IEEE Transactions on Robotics and Automation*, 18(5):665–661, 2002.

K. J. Åström. Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205, 1965.

Y. Aviv and A. Pazgal. A partially observed Markov decision process for dynamic pricing. *Management Science*, 51(9):1400–1416, 2005.

J. A. Bagnell, S. Kakade, A. Y. Ng, and J. Schneider. Policy search by dynamic programming. In *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.

B. Bakker. Reinforcement learning with long short-term memory. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2002.

B. Bakker, F. Linåker, and J. Schmidhuber. Reinforcement learning in partially observable mobile robot domains using unsupervised event extraction. In *Proc. of International Conference on Intelligent Robots and Systems*, 2002.

A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138, 1995.

J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, 22:423–455, 2004.

R. Bellman. *Dynamic programming.* Princeton University Press, 1957.

D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

D. S. Bernstein, E. A. Hansen, and S. Zilberstein. Bounded policy iteration for decentralized POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2005.

D. P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, MA, 2nd edition, 2000.

D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice Hall, 1989.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1996.

D. P. Bertsekas and J. N. Tsitsiklis. *Introduction to Probability.* Athena Scientific, Belmont, MA, 2002.

D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, Belmont, MA, 1997.

J. Blythe. Decision-theoretic planning. *AI Magazine*, 20(2):37–54, 1999.

J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2005.

B. Bonet. An epsilon-optimal grid-based algorithm for partially observable Markov decision processes. In *International Conference on Machine Learning*, 2002.

C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Theoretical Aspects of Rationality and Knowledge*, 1996.

C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 2000.

C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proc. of the National Conference on Artificial Intelligence*, 1996.

R. I. Brafman. A heuristic variable grid solution method for POMDPs. In *Proc. of the National Conference on Artificial Intelligence*, 1997.

D. Braziunas and C. Boutilier. Stochastic local search for POMDP controllers. In *Proc. of the National Conference on Artificial Intelligence*, 2004.

A. Brooks, A. Makarenko, S. Williams, and H. Durrant-Whyte. Planning in continuous state spaces with parametric POMDPs. In *IJCAI Workshop: Reasoning with Uncertainty in Robotics*, 2005.

R. A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1-2):3–15, 1990.

A. R. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes.* PhD thesis, Brown University, 1998a.

A. R. Cassandra. A survey of POMDP applications. In *Working Notes of the AAAI Fall Symposium on Planning with Partially Observable Markov Decision Processes*, 1998b.

A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien. Acting under uncertainty: Discrete Bayesian models for mobile robot navigation. In *Proc. of International Conference on Intelligent Robots and Systems*, 1996.

A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *Proc. of the National Conference on Artificial Intelligence*, 1994.

A. R. Cassandra, M. L. Littman, and N. L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proc. of Uncertainty in Artificial Intelligence*, 1997.

H. T. Cheng. *Algorithms for partially observable Markov decision processes.* PhD thesis, University of British Columbia, 1988.

T. Darrell and A. Pentland. Active gesture recognition using partially observable Markov decision processes. In *Proc. of the 13th Int. Conf. on Pattern Recognition*, 1996.

I. David, L. Friedman, and Z. Sinuany-Stern. A simple suboptimal algorithm for system maintenance under partial observability. *Annals of Operations Research*, 91:25–40, 1999.

A. P. Dempster. A generalization of Bayesian inference. *Journal of the Royal Statistical Society, Series B*, 30:205–247, 1968.

M. B. Dias, A. Stentz, and D. Goldberg. Market-based multirobot coordination for complex space applications. In *Proc. of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2003.

A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo in Practice*. Springer-Verlag, New York, 2001.

A. W. Drake. *Observation of a Markov process through a noisy channel*. Sc.D. thesis, Massachusetts Institute of Technology, 1962.

M. Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachusetts, Amherst, 2002.

E. B. Dynkin. Controlled random sequences. *Theory of probability and its applications*, 10(1):1–14, 1965.

J. N. Eagle. The optimal search for a moving target when the search path is constrained. *Operations Research*, 32(5):1107–1115, 1984.

J. H. Ellis, M. Jiang, and R. Corotis. Inspection, maintenance, and repair with partial observability. *Journal of Infrastructure Systems*, 1(2):92–99, 1995.

R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, 2004.

R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Game theoretic control for robot teams. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2005.

J. A. Feldman and R. F. Sproull. Decision theory and artificial intelligence II: The hungry monkey. *Cognitive Science*, 1(2):125–234, 1977.

Z. Feng and S. Zilberstein. Region-based incremental pruning for POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*, 2004.

D. Ferguson and A. Stentz. Multi-resolution field D*. In *Int. Conf. on Intelligent Autonomous Systems*, 2006.

R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

A. Foka and P. Trahanias. Real-time hierarchical POMDPs for autonomous robot navigation. In *IJCAI Workshop: Reasoning with Uncertainty in Robotics*, 2005.

D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

M. S. Fox, M. Barbuceanu, and R. Teigen. Agent-oriented supply-chain management. *International Journal of Flexible Manufacturing Systems*, 12(2,3), 2000.

P. J. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.

J. Goldberger and S. Roweis. Hierarchical clustering of a mixture model. In *Advances in Neural Information Processing Systems 17*. MIT Press, 2005.

C. V. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, 2003.

C. V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22:143–174, 2004.

C. Guestrin, D. Koller, and R. Parr. Solving factored POMDPs with linear value functions. In *IJCAI-01 workshop on Planning under Uncertainty and Incomplete Information*, 2001.

C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2002a.

C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19: 399–468, 2003.

C. Guestrin, S. Venkataraman, and D. Koller. Context-specific multiagent coordination and planning with factored MDPs. In *Proc. of the National Conference on Artificial Intelligence*, July 2002b.

A. Guo and V. Lesser. Planning for weakly-coupled partially observable stochastic games. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2005.

E. A. Hansen. *Finite-memory control of partially observable systems.* PhD thesis, University of Massachusetts, Amherst, 1998a.

E. A. Hansen. Solving POMDPs by searching in policy space. In *Proc. of Uncertainty in Artificial Intelligence*, 1998b.

E. A. Hansen, D. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proc. of the National Conference on Artificial Intelligence*, 2004.

E. A. Hansen and Z. Feng. Dynamic programming for POMDPs using a factored state representation. In *Int. Conf. on Artificial Intelligence Planning and Scheduling*, 2000.

M. Hauskrecht. Value function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–95, 2000.

M. Hauskrecht and H. Fraser. Planning treatment of ischemic heart disease with partially observable Markov decision processes. *Artificial Intelligence in Medicine*, 18:221–244, 2000.

N. Hernandez-Gardiol and S. Mahadevan. Hierarchical memory-based reinforcement learning. In *Advances in Neural Information Processing Systems 13*. MIT Press, 2001.

J. Hoey and P. Poupart. Solving POMDPs with continuous or large discrete observation spaces. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2005.

J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proc. of Uncertainty in Artificial Intelligence*, 1999.

A. Howard, L. E. Parker, and G. S. Sukhatme. The SDR experience: Experiments with a large-scale heterogeneous mobile robot team. In *Proc. of 9th International Symposium on Experimental Robotics*, 2004.

R. A. Howard. *Dynamic Programming and Markov Processes.* MIT Press and John Wiley & Sons, Inc., 1960.

S.-P. Hsu and A. Arapostathis. Safety control of partially observed MDPs with applications to machine maintenance problems. In *IEEE Int. Conf. on Systems, Man and Cybernetics*, 2004.

C. Hu, W. S. Lovejoy, and S. L. Shafer. Comparison of some suboptimal control policies in medical drug therapy. *Operations Research*, 44(5):696–709, 1996.

T. Huntsberger, P. Pirjanian, A. Trebi-Ollennu, H. Das Nayar, H. Aghazarian, A. J. Ganino, M. Garrett, S. S. Joshi, and P. S. Schenker. CAMPOUT: A control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 33(5):550–559, 2003.

M. R. James, S. Singh, and M. L. Littman. Planning with predictive state representations. In *Proc. of Int. Conf. on Machine Learning and Applications*, 2004.

M. R. James, T. Wessling, and N. Vlassis. Improving approximate value iteration using memories and predictive state representations. In *Proc. of the National Conference on Artificial Intelligence*, 2006.

F. V. Jensen. *Bayesian Networks and Decision Graphs.* Springer-Verlag, 2001.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

M. Kearns, Y. Mansour, and A. Y. Ng. Approximate planning in large POMDPs via reusable trajectories. In *Advances in Neural Information Processing Systems 12.* MIT Press, 2000.

J. R. Kok, M. T. J. Spaan, and N. Vlassis. Multi-robot decision making using coordination graphs. In *Proceedings of the 11th International Conference on Advanced Robotics*, pages 1124–1129, Coimbra, Portugal, 2003.

J. R. Kok, M. T. J. Spaan, and N. Vlassis. Non-communicative multi-robot coordination in dynamic environments. *Robotics and Autonomous Systems*, 50(2-3):99–114, Feb. 2005.

R. E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987.

M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

V. Lesser, C. L. Ortiz, and M. Tambe, editors. *Distributed sensor networks: A multiagent perspective.* Kluwer Academic Publishers, 2003.

A. Likas, N. Vlassis, and J. J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition*, 36(2):451–461, 2003.

Z.-Z. Lin, J. C. Bean, and C. C. White. A hybrid genetic/optimization algorithm for finite horizon, partially observed Markov decision processes. *INFORMS Journal on Computing*, 16(1):27–38, 2004.

M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *International Conference on Machine Learning*, 1995.

M. L. Littman, R. S. Sutton, and S. Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2002.

W. S. Lovejoy. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, 39(1):162–175, 1991.

R. D. Luce and H. Raiffa. *Games and decisions: introduction and critical survey.* Wiley, 1957.

O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1-2):5–34, 2003.

N. Meuleau, K.-E. Kim, L. P. Kaelbling, and A. R. Cassandra. Solving POMDPs by searching the space of finite policies. In *Proc. of Uncertainty in Artificial Intelligence*, 1999a.

N. Meuleau, L. Peshkin, K.-E. Kim, and L. P. Kaelbling. Learning finite-state controllers for partially observable environments. In *Proc. of Uncertainty in Artificial Intelligence*, 1999b.

P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, 2003.

G. E. Monahan. A survey of partially observable Markov decision processes: theory, models and algorithms. *Management Science*, 28(1), Jan. 1982.

A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.

K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning.* PhD thesis, University of California, Berkeley, 2002.

R. Nair, M. Tambe, M. Roth, and M. Yokoo. Communication for improving policy computation in distributed POMDPs. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, 2004.

R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2003.

R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proc. of the National Conference on Artificial Intelligence*, 2005.

A. Y. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*, 2000.

J. M. Ooi and G. W. Wornell. Decentralized control of a multiple access broadcast channel: Performance bounds. In *Proc. of the 35th Conference on Decision and Control*, 1996.

C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1993.

C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

A. Papoulis. *Probability, random variables and stochastic processes.* McGraw-Hill, 3rd edition, 1991.

S. Paquet, L. Tobin, and B. Chaib-draa. An online POMDP algorithm for complex multiagent environments. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, 2005.

J. Peng and R. J. Williams. Efficient learning and planning within the Dyna framework. *Adaptive Behavior*, 1(4):437–454, 1993.

L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling. Learning to cooperate via policy search. In *Proc. of Uncertainty in Artificial Intelligence*, 2000.

J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2003a.

J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun. Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, 42(3–4):271–281, 2003b.

J. Pineau and S. Thrun. An integrated approach to hierarchy and abstraction for POMDPs. Technical Report CMU-RI-TR-02-21, Robotics Institute, Carnegie Mellon University, 2002.

L. K. Platzman. A feasible computational approach to infinite-horizon partially-observed Markov decision problems. Technical Report J-81-2, School of Industrial and Systems Engineering, Georgia Institute of Technology, 1981. Reprinted in working notes AAAI 1998 Fall Symposium on Planning with POMDPs.

K.-M. Poon. A fast heuristic algorithm for decision-theoretic planning. Master's thesis, The Hong-Kong University of Science and Technology, 2001.

J. M. Porta and B. J. A. Kröse. Appearance-based concurrent map building and localization using a multi-hypotheses tracker. In *Proc. of International Conference on Intelligent Robots and Systems*, 2004.

J. M. Porta, M. T. J. Spaan, and N. Vlassis. Value iteration for continuous-state POMDPs. Technical Report IAS-UVA-04-04, Informatics Institute, University of Amsterdam, Dec. 2004.

J. M. Porta, M. T. J. Spaan, and N. Vlassis. Robot planning in partially observable continuous domains. In *Robotics: Science and Systems*, pages 217–224. MIT Press, 2005.

P. Poupart. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes.* PhD thesis, University of Toronto, 2005.

P. Poupart and C. Boutilier. Value-directed compression of POMDPs. In *Advances in Neural Information Processing Systems 15.* MIT Press, 2003a.

P. Poupart and C. Boutilier. Value-directed compression of POMDPs. In *Advances in Neural Information Processing Systems 15.* MIT Press, 2003b.

P. Poupart and C. Boutilier. Bounded finite state controllers. In *Advances in Neural Information Processing Systems 16.* MIT Press, 2004.

P. Poupart and C. Boutilier. VDCBPI: an approximate scalable algorithm for large scale POMDPs. In *Advances in Neural Information Processing Systems 17.* MIT Press, 2005.

P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete Bayesian reinforcement learning. In *International Conference on Machine Learning*, 2006.

M. L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, 1994.

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

S. M. Ross. Quality control under Markovian deterioration. *Management Science*, 17(9):587–596, 1971.

M. Roth. Execution-time communication decisions for coordination of multiagent teams. Thesis proposal. Robotics Institute, Carnegie Mellon University, Jan. 2005.

M. Roth, R. Simmons, and M. Veloso. Decentralized communication strategies for coordinated multi-agent policies. In A. Schultz, L. Parker, and F. Schneider, editors, *Multi-Robot Systems: From Swarms to Intelligent Automata*, volume IV. Kluwer Academic Publishers, 2005.

N. Roy. *Finding Approximate POMDP Solutions Through Belief Compression.* PhD thesis, Carnegie Mellon University, 2003.

N. Roy and G. Gordon. Exponential family PCA for belief compression in POMDPs. In *Advances in Neural Information Processing Systems 15.* MIT Press, 2003.

N. Roy, G. Gordon, and S. Thrun. Planning under uncertainty for reliable health care robotics. In *Proc. of the Int. Conf. on Field and Service Robotics*, 2003.

N. Roy, G. Gordon, and S. Thrun. Finding approximate POMDP solutions through belief compression. *Journal of Artificial Intelligence Research*, 23: 1–40, 2005.

N. Roy, J. Pineau, and S. Thrun. Spoken dialog management for robots. In *Proc. of the Association for Computational Linguistics*, 2000.

N. Roy and S. Thrun. Coastal navigation with mobile robots. In *Advances in Neural Information Processing Systems 12.* MIT Press, 2000.

P. Rusmevichientong and B. Van Roy. A tractable POMDP for a class of sequencing problems. In *Proc. of Uncertainty in Artificial Intelligence*, 2001.

S. J. Russell and P. Norvig. *Artificial Intelligence: a modern approach.* Prentice Hall, 2nd edition, 2003.

H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, 1984.

J. K. Satia and R. E. Lave. Markovian decision processes with probabilistic observation of states. *Management Science*, 20(1), 1973.

S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. In *Proc. of the National Conference on Artificial Intelligence*, 1994.

G. Shafer. *A Mathematical Theory of Evidence.* Princeton University Press, 1976.

G. Shani, R. Brafman, and S. Shimony. Model-based online learning of POMDPs. In *European Conference on Machine Learning*, 2005a.

G. Shani, R. I. Brafman, and S. E. Shimony. Adaptation for changing stochastic environments through online POMDP policy learning. In *ECML Workshop on Reinforcement Learning in Non-Stationary Environments*, 2005b.

R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 1995.

J. Simmons Ivy and H. Black Nembhard. A modeling approach to maintenance decisions using statistical quality control and optimization. *Quality and Reliability Engineering International*, 21:355–366, 2005.

S. Singh, T. Jaakkola, and M. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *International Conference on Machine Learning*, 1994.

S. Singh, M. R. James, and M. R. Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *Proc. of Uncertainty in Artificial Intelligence*, 2004.

R. D. Smallwood and E. J. Sondik. The optimal control of partially observable Markov decision processes over a finite horizon. *Operations Research*, 21: 1071–1088, 1973.

T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*, 2004.

E. J. Sondik. *The optimal control of partially observable Markov processes.* PhD thesis, Stanford University, 1971.

E. J. Sondik. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, 1978.

H. W. Sorenson, editor. *Kalman filtering: theory and application.* IEEE Press, 1985.

M. T. J. Spaan, G. J. Gordon, and N. Vlassis. Decentralized planning under uncertainty for teams of communicating agents. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 249–256, 2006.

M. T. J. Spaan and N. Vlassis. A point-based POMDP algorithm for robot planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2399–2404, New Orleans, Louisiana, 2004.

M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005a.

M. T. J. Spaan and N. Vlassis. Planning with continuous actions in partially observable environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3469–3474, Barcelona, Spain, 2005b.

R. L. Stratonovich. Conditional Markov processes. *Theory of probability and its applications*, 5(2):156–178, 1960.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

J. M. Swaminathan, S. F. Smith, and N. M. Sadeh. Modeling supply chain dynamics: A multiagent approach. *Decision Sciences*, 29(3), 1998.

C. Szepesvári and M. L. Littman. Generalized Markov decision processes: Dynamic-programming and reinforcement-learning algorithms. Technical Report CS-96-11, Brown University, Department of Computer Science, Nov. 1996.

D. Szer and F. Charpillet. An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs. In *European Conference on Machine Learning*, 2005.

M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *International Conference on Machine Learning*, 1993.

N. Tao, J. Baxter, and L. Weaver. A multi-agent policy-gradient approach to network routing. In *International Conference on Machine Learning*, 2001.

G. Theocharous and S. Mahadevan. Approximate planning with hierarchical partially observable Markov decision processes for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002.

G. Theocharous, K. Murphy, and L. P. Kaelbling. Representing hierarchical POMDPs as DBNs for multi-scale robot localization. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2004.

S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

S. Thrun. Monte Carlo POMDPs. In *Advances in Neural Information Processing Systems 12.* MIT Press, 2000.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics.* MIT Press, 2005.

J. T. Treharne and C. R. Sox. Adaptive inventory control for nonstationary demand and partial information. *Management Science*, 48(5):607–624, 2002.

E. Tuttle and Z. Ghahramani. Propagating uncertainty in POMDP value iteration with Gaussian processes. Technical report, Gatsby Computational Neuroscience Unit, Aug. 2004.

P. Varakantham, R. Maheswaran, and M. Tambe. Exploiting belief bounds: Practical POMDPs for personal assistant agents. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, 2005.

N. Vlassis and M. T. J. Spaan. A fast point-based algorithm for POMDPs. In *Benelearn 2004: Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands*, pages 170–176, Brussels, Belgium, Jan. 2004. (Also presented at the NIPS 16 workshop 'Planning for the Real-World', Whistler, Canada, Dec 2003).

N. Vlassis, B. Terwijn, and B. J. A. Kröse. Auxiliary particle filter robot localization from high-dimensional sensor observations. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002.

N. Vlassis and J. J. Verbeek. Gaussian mixture learning from noisy data. Technical Report IAS-UVA-04-01, Informatics Institute, University of Amsterdam, Sept. 2004.

T. Wang, D. Lizotte, M. Bowling, and D. Schuurmans. Bayesian sparse sampling for on-line reward optimization. In *International Conference on Machine Learning*, 2005.

C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, 1989.

C. J. C. H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

C. C. White. A Markov quality control process subject to partial observation. *Management Science*, 23(8):843–852, 1977.

C. C. White. Partially observed Markov decision processes: a survey. *Annals of Operations Research*, 32, 1991.

S. D. Whitehead and L.-J. Lin. Reinforcement learning of non-Markov decision processes. *Artificial Intelligence*, 73(1-2):271–306, 1995.

M. Wiering and J. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6(2):219–246, 1997.

J. D. Williams, P. Poupart, and S. Young. Factored partially observable Markov decision processes for dialogue management. In *4th IJCAI Workshop on Knowledge and Reasoning in Practical Dialog Systems*, 2005a.

J. D. Williams, P. Poupart, and S. Young. Partially observable Markov decision processes with continuous observations for dialogue management. In *Proceedings of the 6th SigDial Workshop on Discourse and Dialogue*, 2005b.

J. D. Williams and S. Young. Scaling up POMDPs for dialog management: The "Summary POMDP"method. In *IEEE Automatic Speech Recognition and Understanding Workshop*, 2005.

P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proc. of the Fifth Int. Conference on Autonomous Agents*, 2001.

L. Zadeh. Fuzzy sets as the basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.

N. L. Zhang and W. Liu. Planning in stochastic domains: problem characteristics and approximations. Technical Report HKUST-CS96-31, Department of Computer Science, The Hong Kong University of Science and Technology, 1996.

N. L. Zhang and W. Zhang. Speeding up the convergence of value iteration in partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 14:29–51, 2001.

R. Zhou and E. A. Hansen. An improved grid-based approximation algorithm for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2001.

Z. Zivkovic, B. Bakker, and B. J. A. Kröse. Hierarchical map building using visual landmarks and geometric constraints. In *Proc. of International Conference on Intelligent Robots and Systems*, 2005.

Publications

This thesis is based on adaptations of the following publications:

M. T. J. Spaan and N. Vlassis. A point-based POMDP algorithm for robot planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2399–2404, 2004. Chapter 3.

M. T. J. Spaan and N. Vlassis. Planning with continuous actions in partially observable environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3469–3474, 2005. Chapter 4.

J. M. Porta, M. T. J. Spaan, and N. Vlassis. Robot planning in partially observable continuous domains. In *Robotics: Science and Systems*, pages 217–224. MIT Press, 2005. Chapter 4.

M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005. Chapters 3 and 4.

M. T. J. Spaan, G. J. Gordon, and N. Vlassis. Decentralized planning under uncertainty for teams of communicating agents. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 249–256, 2006. Chapter 5.

Other recent international publications:

J. R. Kok, M. T. J. Spaan, and N. Vlassis. Multi-robot decision making using coordination graphs. In *Proceedings of the 11th International Conference on Advanced Robotics*, pages 1124–1129, 2003.

J. R. Kok, M. T. J. Spaan, and N. Vlassis. Non-communicative multi-robot coordination in dynamic environments. *Robotics and Autonomous Systems*, 50(2-3):99–114, Feb. 2005.

Acknowledgments

I would like to thank Frans Groen for supporting me ever since I joined his group, almost six years ago. I have learnt a lot during all these years. Many thanks go to Nikos Vlassis for supervising my PhD, keeping me focused, and sharing his insights during this very pleasant collaboration. He was always able to put me back on track when an idea did not work out as I had planned.

I am also grateful for the fruitful cooperation with Josep M Porta and Geoff Gordon. Thanks to Geoff for hosting me in Pittsburgh for a great three-month visit.

It was my fortune to be able to share the day-to-day joys and troubles of life as a PhD student with roommates Jelle Kok and Bram Bakker, and all other (former) group members. Thanks for listening. I've enjoyed the Sunday afternoon sessions of Emergentia, with the correct balance between science and leisure.

There's more to life than science, so many thanks to all friends who have actively contributed to my social well-being throughout these years. Furthermore, I am grateful to my parents and sister for their continuing love, support and encouragement.