Chapter 4

Factored Markov Decision Processes

4.1. Introduction

Solution methods described in the MDP framework (Chapters 1 and2) share a common bottleneck: they are not adapted to solve large problems. Indeed, using non structured representations requires an explicit enumeration of the possible states in the problem. The complexity of this enumeration grows exponentially with the number of variables in the problem.

EXAMPLE.– In the case of the car to be maintained, the number of possible states of a car can be huge. For instance, each part of the car can have its one wearout state. The idea of factored representations is that some part of this huge state do not depend on each other and that this structure can be exploited to derive a more compact representation of the global state and obtain more efficiently an optimal policy. For instance, changing the oil in the car should have no effect on the breaks, thus one does not need to care about the state of the breaks to determine an optimal oil changing policy.

This chapter aims to describe FMDPs (Factored Markov Decision Processes), first proposed by [BOU 95, BOU 99]. FMDPs are an extension of MDPs that allows to represent the transition and the reward functions of some problems compactly (compared to an explicit enumeration of state-action pairs). First, we describe the framework and how problems are modeled (Section 4.2). Then we describe different planning methods able to take advantage of the structure of the problem to compute optimal or near-optimal solutions (Section 4.3). Finally, we conclude in Section 4.4 and present some perspectives.

Chapter written by Thomas DEGRIS and Olivier SIGAUD.

4.2. Modeling a Problem with an FMDP

4.2.1. *Representing the State Space*

It is intuitive to describe a problem by a set of observations whose values describe the current status of the environment. So, the state s can be described as a multivariate random variable $\boldsymbol{X} = (X_1, \ldots, X_n)$ where each variable X_i can take a value in its domain $\text{DOM}(X_i)$.[1] Then, a state becomes an instantiation of each random variable X_i and can be written as a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ such that $\forall i \; x_i \in \text{DOM}(X_i)$. We note $\text{DOM}(\boldsymbol{X})$ the set of possible instantiations for the multivariate variable $\boldsymbol{X}$. Consequently, the state space S of the MDP is defined by $S = \text{DOM}(\boldsymbol{X})$.

With such representations, states are not atomic and it becomes possible to exploit some structures of the problem. In FMDPs, such representation is mainly used to represent the problem compactly and to reduce the complexity of the computation of the solution. More precisely, FMDPs exploit *function-specific independence* to represent compactly the transition and the reward functions. Moreover, FMDPs are also appropriate to exploit two other properties related to the structure of the problem, that is *context-specific independence* and linear approximation.

To illustrate the FMDP framework, we are going to use a well known example in the literature named *Coffee Robot* (Section 4.2.2), first proposed by [BOU 00]. Using this example, we are then going to describe the decomposition of the transition and the reward functions (Section 4.2.3) with a formalization of function-specific independencies. Section 4.2.4 proposes a formalization of context-specific independencies.

4.2.2. *The* Coffee Robot *Example*

A robot must go to a café to buy a cup of coffee for its owner who is located at its office. When it is raining, it must get an umbrella to stay dry when going to the café. The state of the system is composed of six binary variables[2] X_i where $\text{DOM}(X_i) = \{0, 1\}$ (corresponding respectively to `False` and `True`). These variables are:

$\mathcal{H}$: Has the owner a coffee?

$\mathcal{C}$: Has the robot a coffee?

---

1. For convenience, we will sometimes see such multivariate random variables as sets of random variables.

2. This example uses binary variables. However, nothing prevents from using a random variable X_i where $|\text{DOM}(X_i)| > 2$.

$\mathcal{W}$: Is the robot wet?

$\mathcal{R}$: Is it raining?

$\mathcal{U}$: Has the robot an umbrella?

$\mathcal{O}$: Is the robot in the office?

For instance, the vector $[\mathcal{H}{=}0,\mathcal{C}{=}1,\mathcal{W}{=}0,\mathcal{R}{=}1,\mathcal{U}{=}0,\mathcal{O}{=}1]$ represents the state of the *Coffee Robot* problem where the owner does not have a coffee, the robot has a coffee, it is not wet, it is raining, the robot does not have an umbrella and it is located in the office. This problem being composed of 6 binary variables, there is $2^6 = 64$ possible states.

In this problem, four actions are available to the robot:

$\mathcal{G}$o: Move to the other location.

$\mathcal{B}$uyC: Buy a coffee: the robot gets one only if it is in the café.

$\mathcal{D}$elC: Deliver coffee: the owner will get a coffee only if the robot is located in the office and it has a coffee.

$\mathcal{G}$etU: Get an umbrella: the robot will get one only if it is located in the office.

Actions can be noisy to represent stochastic problems. For instance, when the robot gives the coffee, its owner will get his coffee only with a given probability (the cup may fall). Thus, when the action $\mathcal{D}$elC is executed in the state $\boldsymbol{x} = [\mathcal{C}{=}0,\mathcal{H}{=}1,\mathcal{W}{=}0,\mathcal{R}{=}1,\mathcal{U}{=}0,\mathcal{O}{=}1]$ (the robot is in the office and the owner does not have a coffee), the transition function of the problem defines:

- $P([\mathcal{C}{=}1,\mathcal{H}{=}1,\mathcal{W}{=}0,\mathcal{R}{=}1,\mathcal{U}{=}0,\mathcal{O}{=}1]|\boldsymbol{x}, \mathcal{D}\text{elC}) = 0.8$,
- $P([\mathcal{C}{=}0,\mathcal{H}{=}1,\mathcal{W}{=}0,\mathcal{R}{=}1,\mathcal{U}{=}0,\mathcal{O}{=}1]|\boldsymbol{x}, \mathcal{D}\text{elC}) = 0.2$,
- 0.0 for other transition probabilities.

Finally, for the reward function, the robot gets a reward of 0.9 when the owner has a coffee (0 when it does not) and 0.1 when it is dry (and 0 when the robot is wet). The reward the robot obtains when the owner gets a coffee is larger than the reward obtained when the robot is dry so as to specify that the task of getting a coffee has a higher priority than the constraint of staying dry.

4.2.3. *Decomposition and Function-Specific Independence*

Function-specific independencies refer to the property that some functions in the problem do not depend on all the variables in the problem or on the action executed by the agent. For instance, in the *Coffee Robot* problem, the value of the variable $\mathcal{R}$ at the next time step, meaning if it is raining or not, only depends on its own value at the current time step. Indeed, the fact that it is going to rain at the next time step is independent of variables such as "Has the robot a coffee ?" (variable $\mathcal{H}$) or the last action executed by the robot.

The FMDP framework allows to take advantage of such independencies in the representation of the transition and reward functions. Once these independencies specified, planning algorithms can then exploit them to improve their complexity. This notion is formalized by two different operators, namely PARENTS and SCOPE, defined respectively in the next section (Section 4.2.3.1) for the transition function and in Section 4.2.3.4 for the reward function.

4.2.3.1. *Transition Function Decomposition*

A transition function in a finite MDP is defined by the finite set of probabilities $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, a_t)$. Assuming that the state can be decomposed using multiple random variables (see Section 4.2.1), it is possible to decompose the probability $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, a_t)$ as a product of probabilities, then to exploit the independencies between these random variables to decrease the size of the representation of the transition function.

For instance, in the *Coffee Robot* problem, the transition function for the action $\mathcal{D}\text{elC}$ is defined by the set of probabilities $P_{\mathcal{D}\text{elC}}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)$. By first decomposing the state $\boldsymbol{x}_{t+1}$ and using the Bayes rule, we have:

$$\begin{aligned} P_{\mathcal{D}\text{elC}}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t) &= P_{\mathcal{D}\text{elC}}(c_{t+1}, h_{t+1}, w_{t+1}, r_{t+1}, u_{t+1}, o_{t+1}|\boldsymbol{x}_t) \\ &= P_{\mathcal{D}\text{elC}}(c_{t+1}|\boldsymbol{x}_t) * P_{\mathcal{D}\text{elC}}(h_{t+1}|\boldsymbol{x}_t, c_{t+1}) * \ldots * \\ &\quad P_{\mathcal{D}\text{elC}}(o_{t+1}|\boldsymbol{x}_t, c_{t+1}, h_{t+1}, w_{t+1}, r_{t+1}, u_{t+1}) \end{aligned}$$

where $\boldsymbol{X}_t$ is a random variable representing the state at time t. In the case of the *Coffee Robot* problem, we know that the value of a variable X_{t+1} only depends on the variables at time t, so that we have:

$$\begin{aligned} P_{\mathcal{D}\text{elC}}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t) &= P_{\mathcal{D}\text{elC}}(c_{t+1}|\boldsymbol{x}_t) * P_{\mathcal{D}\text{elC}}(h_{t+1}|\boldsymbol{x}_t, c_{t+1}) * \ldots * \\ &\quad P_{\mathcal{D}\text{elC}}(o_{t+1}|\boldsymbol{x}_t, c_{t+1}, h_{t+1}, w_{t+1}, r_{t+1}, u_{t+1}) \\ &= P_{\mathcal{D}\text{elC}}(c_{t+1}|\boldsymbol{x}_t) * \ldots * P_{\mathcal{D}\text{elC}}(o_{t+1}|\boldsymbol{x}_t). \end{aligned}$$

Similarly, it is possible to decompose the state $\boldsymbol{x}_t$:

$$\begin{aligned} P_{\mathcal{D}\text{elC}}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t) &= P_{\mathcal{D}\text{elC}}(c_{t+1}|\boldsymbol{x}_t) * \ldots * P_{\mathcal{D}\text{elC}}(o_{t+1}|\boldsymbol{x}_t) \\ &= P_{\mathcal{D}\text{elC}}(c_{t+1}|c_t, h_t, w_t, r_t, u_t, o_t) * \ldots * \\ &\quad P_{\mathcal{D}\text{elC}}(o_{t+1}|c_t, h_t, w_t, r_t, u_t, o_t). \end{aligned}$$

Then, given the structure of the *Coffee Robot* problem, we know, for instance, that the probability $P_{\mathcal{D}\text{elC}}(\mathcal{R}_{t+1} = 1|\boldsymbol{x}_t)$ that it will rain at the next time step only depends on r_t, that is if it is raining at time t. Thus, we have $P_{\mathcal{D}\text{elC}}(\mathcal{R}_{t+1} = 1|\boldsymbol{x}_t) = P_{\mathcal{D}\text{elC}}(\mathcal{R}_{t+1} = 1|r_t)$. By exploiting function-specific independencies, it is possible to compactly represent the transition function for the action $\mathcal{D}\text{elC}$ by defining each probability $P_{\mathcal{D}\text{elC}}(x_{t+1}|\boldsymbol{x}_t)$ only by the variables it depends on at time t in the problem (rather than all the variables composing the state $\boldsymbol{x}_t$):

$$\begin{aligned} P_{\mathcal{D}\text{elC}}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t) &= P_{\mathcal{D}\text{elC}}(c_{t+1}|c_t, h_t, w_t, r_t, u_t, o_t) * \ldots * \\ &\quad P_{\mathcal{D}\text{elC}}(o_{t+1}|c_t, h_t, w_t, r_t, u_t, o_t) \\ &= P_{\mathcal{D}\text{elC}}(c_{t+1}|c_t, h_t, o_t) * P_{\mathcal{D}\text{elC}}(h_{t+1}|h_t, o_t) * \\ &\quad P_{\mathcal{D}\text{elC}}(w_{t+1}|w_t) * P_{\mathcal{D}\text{elC}}(u_{t+1}|u_t) * \\ &\quad P_{\mathcal{D}\text{elC}}(r_{t+1}|r_t) * P_{\mathcal{D}\text{elC}}(o_{t+1}|o_t). \end{aligned}$$

So, for instance, rather than defining the probability distribution $P_{\mathcal{D}\text{elC}}(\mathcal{R}_{t+1}|\boldsymbol{X}_t)$, we now use only the required variables, that is the same variable at time t: $P_{\mathcal{D}\text{elC}}(\mathcal{R}_{t+1}|\boldsymbol{X}_t) = P_{\mathcal{D}\text{elC}}(\mathcal{R}_{t+1}|\mathcal{R}_t)$. The *Coffee Robot* problem is composed of six binary variables, meaning that it requires to define $2^6 * 2^6 = 4096$ probabilities $P_{\mathcal{D}\text{elC}}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)$ for the action $\mathcal{D}\text{elC}$ whereas the compact representation above only requires to define $2 * 2^3 + 2 * 2^2 + 2 * 2^1 + 2 * 2^1 + 2 * 2^1 + 2 * 2^1 = 40$ probabilities.

Consequently, function-specific independence related to the structure of the problem are explicitly defined and allow to aggregate regularities in the transition function. Moreover, function-specific independence refers to an often intuitive representation of a problem by describing the consequences of actions on the values of the different variables. In FMDPs, function-specific independencies are formalized with dynamic Bayesian networks [BOU 95].

4.2.3.2. *Dynamic Bayesian Networks in FMDPs*

Bayesian networks [PEA 88] are a representational framework to represent dependencies (or independencies) between random variables. These variables are the nodes of a directed graph. Direct probabilistic dependencies between two variables are represented by an edge between the two nodes representing these two variables. Dynamic Bayesian networks (DBNs) are Bayesian networks representing temporal stochastic processes. The nodes in a DBN represent variables for a particular time slice.

Assuming that the problem is stationary (the transition function T of the MDP does not depend on the time), it is possible to represent T with DBNs using only two successive time steps (assuming the Markov property is satisfied). In such a case, DBNs are composed of two sets of nodes:

1) the set of nodes representing the variables of the problem at time t,

2) the set of nodes representing the variables of the problem at time $t+1$.

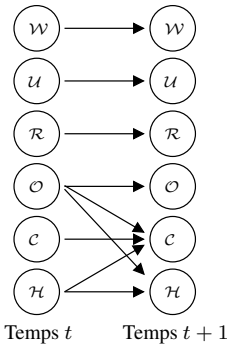
Edges indicate direct dependencies between variables at time t and variables at time $t+1$ or between variables in the same time slice at time $t+1$ (such edges are named *synchronous arcs*). Such DBNs are sometimes named *2 Time-slice Bayesian Networks*.

Independencies between random variables to define the transition function can then be represented using one DBN per action. Similarly, even if the action is rather a decision variables, actions that were executed in the past by the agent can also be considered as a random variable at time t. In such a case, only one DBN is necessary to define independencies between variables (and the action) in the transition function [BOU 96]. Finally, the DBN is quantified by a conditional probability distribution to define the probability of each value $x \in \text{DOM}(X)$ for each variable X given the value of the random variables X directly depends on (its parent variables), as illustrated in the next section with the *Coffee Robot* example.

4.2.3.3. *Factored Model of the Transition Function in an FMDP*

Figure 4.1 represents the effect of the action $\mathcal{D}\text{elC}$ on a state. The DBN $\tau_{\mathcal{D}\text{elC}}$ (Figure 4.1(a)) clearly states that, for instance, for the action $\mathcal{D}\text{elC}$, the variable $\mathcal{C}$ does only depend on the values of the variables $\mathcal{O}$, $\mathcal{H}$ and $\mathcal{C}$ at the previous time step and is independent of the other state variables.

We can define $\text{PARENTS}_\tau(X'_i)$ the set of parents of the variable X'_i in DBN τ. This set can be partitioned in two subsets $\text{PARENTS}^t_\tau(X'_i)$ and $\text{PARENTS}^{t+1}_\tau(X'_i)$ representing respectively the set of parents at time t and the set of parents at time $t+1$. In the *Coffee Robot* example, we assume that there is no synchronous arcs, that is $\text{PARENTS}^{t+1}_\tau(X'_i) = \emptyset$ and $\text{PARENTS}_\tau(X'_i) = \text{PARENTS}^t_\tau(X'_i)$. Thus, in Figure 4.1, we have $\text{PARENTS}_{\mathcal{D}\text{elC}}(\mathcal{C}') = \{\mathcal{O}, \mathcal{H}, \mathcal{C}\}$.



(a)

$\mathcal{C}$	$\mathcal{H}$	$\mathcal{O}$	$\mathcal{C}'$
1	1	1	1.0
1	1	0	1.0
1	0	1	1.0
1	0	0	1.0
0	1	1	0.8
0	1	0	0.0
0	0	1	0.0
0	0	0	0.0

(b)

Figure 4.1. *Partial representation of the transition function T for the* Coffee Robot *problem. Figure (a) represents the dependencies between variables for the $\mathcal{D}$elC action. Figure (b) defines the conditional probability distribution $P_{\mathcal{D}\mathrm{elC}}(\mathcal{C}'|\mathcal{O},\mathcal{H},\mathcal{C})$ using a tabular representation.*

corresponding DBN τ is quantified by a set of conditional probability distributions, noted $P_\tau(X'_i|\textsc{Parents}_\tau(X'_i))$ for a variable X'_i. Then, the probability $P_\tau(\boldsymbol{X}'|\boldsymbol{X})$ can be defined compactly as:

$$P_\tau(\boldsymbol{x}'|\boldsymbol{x}) = \prod_i P_\tau(x'_i|parents(x'_i)) \tag{4.1}$$

with x'_i the value of the variable X'_i in state $\boldsymbol{x}'$ and $parents(x'_i)$ the values of the variables in the set $\textsc{Parents}_\tau(X'_i)$.

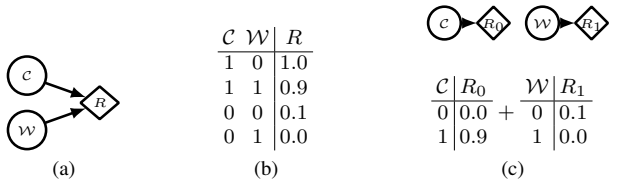
Figure 4.1(b) gives the conditional probability distribution $P_{\mathcal{D}\mathrm{elC}}(\mathcal{C}'|\mathcal{O},\mathcal{H},\mathcal{C})$ in the *Coffee Robot* problem in a tabular form. The columns $\mathcal{O}$, $\mathcal{H}$ and $\mathcal{C}$ represent the values of these variables at time t. The column $\mathcal{C}'$ represents the probability for variable $\mathcal{C}$ to be true at time $t+1$.

The multiplicative decomposition (Equation 4.1) and the specification of functional independencies in the model description of the transition function are the main contributions of FMDPs compared to MDPs. Both of these properties are exploited by the algorithms exploiting the structure of the problem specified in the FMDP.

4.2.3.4. *Factored Model of the Reward Function*

A similar representation can be used to specify the reward function of the problem in FMDPs. Indeed, first, the reward function R can be decomposed additively and, second, the different terms of the decomposition do not necessarily depend on all the state variables of the problem.

For instance, in the *Coffee Robot* problem, the reward function, represented by a diamond in Figure 4.2, only depends on two variables $\mathcal{C}$ and $\mathcal{W}$. It is independent of the action executed by the agent or of other variables in the problem.



(a) (b) (c)

Figure 4.2. *Representation of the reward function $R(s)$ in the* Coffee Robot *problem.*

The table in Figure 4.2(b) specifies that the best state for the robot is when its owner has a coffee and the robot is dry whereas the worst case is when its owner does not have a coffee and the robot is wet. We can notice that a preference is given to the state where the owner has a coffee and the robot is dry over the state where the owner does not have a coffee and the robot is dry.

[BOU 00] defines the reward function of the *Coffee Robot* problem by summing the two criteria of the problem: "the owner has a coffee" and "the robot is dry". However, these two criteria are independent of each other. To be able to exploit this additive decomposition of the reward function, [GUE 03b] proposes to represent the reward function of an FMDP as a sum of different *localized reward functions*.

Given the *Coffee Robot* problem, we can define the reward function as being the sum of two localized reward functions depending respectively on the variables $\mathcal{C}$ and $\mathcal{W}$, representing the criteria "the owner has a coffee" and "the robot is dry".

[GUE 03b] formalizes such a structure by first defining the *scope* of a localized function f (noted $\text{SCOPE}(f)$). Similarly to PARENTS for DBNs, the scope of f is defined as follows:

DEFINITION 4.1.– *Scope*
A function f has a scope $\text{SCOPE}(f) = \boldsymbol{C} \subseteq \boldsymbol{X}$ *if* $f : \text{DOM}(\boldsymbol{C}) \rightarrow \mathbb{R}$.

Let a function f such as $\text{SCOPE}(f) = \boldsymbol{C}$, we note $f(x)$ as a shorthand for $f(x[\boldsymbol{C}])$ where $x[\boldsymbol{C}]$ is the restriction of $\boldsymbol{x}$ to the variables in $\boldsymbol{C}$. Consequently, the SCOPE definition allows to define the function-specific independence of f.

It is now possible to define a *localized reward function*. Let a set of localized reward functions $R_1^a, \ldots, R_r^a$ with scope $\text{SCOPE}(R_i^a)$ for each R_i^a constrained to a

subset $C_i^a \subseteq \{X_1, \ldots, X_n\}$, then the reward function associated to action a is defined as:

$$R^a(\boldsymbol{x}) = \sum_{i=1}^{r} R_i^a(\boldsymbol{x}[C_i^a]) \tag{4.2}$$

$$= \sum_{i=1}^{r} R_i^a(\boldsymbol{x}). \tag{4.3}$$

Regarding the *Coffee Robot* example, the problem can be defined by the two reward functions R_1 and R_2 given in Figure 4.2(c) and representing respectively the two criteria "the owner has a coffee" and "the robot is dry" with $\text{SCOPE}(R_1) = \{\mathcal{C}\}$ and $\text{SCOPE}(R_2) = \{\mathcal{W}\}$. We note $R_1(\boldsymbol{x})$ as a shorthand for $R_1(\boldsymbol{x}[\mathcal{C}])$ with $\boldsymbol{x}[\mathcal{C}]$ representing the value of $\mathcal{C}$ in $\boldsymbol{x}$.

Whereas all algorithms in the FMDP framework exploit function-specific independencies of the reward function, they do not necessarily exploit its additive decomposition. Moreover, not all problems exhibit such structure in their reward function.

4.2.4. *Context-Specific Independence*

For a given function and a given context, it is not necessarily required to test every variable on which the function depends to define the output of the function. Such property is named *context-specific independence*.

For the *Coffee Robot* problem, in the definition of the conditional probability distribution $P_{\mathcal{D}\text{elC}}(\mathcal{C}'|\mathcal{O}, \mathcal{H}, \mathcal{C})$ (see Figure 4.1(b)), whereas $\text{PARENTS}_{\mathcal{D}\text{elC}}(\mathcal{C}') = \{\mathcal{O}, \mathcal{H}, \mathcal{C}\}$, it is not necessary to test variables $\mathcal{O}$ and $\mathcal{H}$ if $\mathcal{C} = 1$ when $\mathcal{C}_t = 1$ to know the probability distribution of the variable $\mathcal{C}'$.

From [GUE 03a], a context is formalized as follow:

DEFINITION 4.2.– *Context*
Let a function $f : \boldsymbol{X} \rightarrow \boldsymbol{Y}$. A context $\boldsymbol{c} \in \text{DOM}(\boldsymbol{C})$ is an instantiation of a multivariate random variable $\boldsymbol{C} = (C_0, \ldots, C_j)$ such that $\boldsymbol{C} \subseteq \boldsymbol{X}$. It is noted: $(C_0 = c_0) \wedge \ldots \wedge (C_j = c_j)$ or $C_0 = c_0 \wedge \ldots \wedge C_j = c_j$.

Unlike function-specific independence, exploiting context-specific independence is directly related to the data structure used by the algorithms to solve the problem.

Indeed, the operators PARENTS and SCOPE, representing function-specific independence, define the set of variables on which the function depends. As seen in Section 4.2.3, such structure allows to compactly represent some problems, even when the data structure used to define these functions is not structured, as it is the case for the tabular representation used in Figure 4.1.

For a given set of variables (specified with the PARENTS and SCOPE operators), context-specific independencies are used to represent a function more compactly. In this case, the main idea is to use structured representations to aggregate similar states, unlike tabular representations. Thus, [BOU 00] suggests different data structures to represent the different functions of a given FMDP, such as: rules [POO 97], decision lists [RIV 87] or algebraic decision diagrams [BRY 86]. Consequently, because each algorithm proposed in the FMDP framework uses a different data structure and strongly relies on it, we have preferred focusing on the description of these data structures in the next section.

4.3. Planning with FMDPs

This section describes different planning methods to solve problems specified as FMDPs. Rather than describing the algorithms in details, we describe the different representations and data structures they use as an outline of their main properties. However, all the required references are given for the reader interested in more in-depth descriptions.

4.3.1. *Structured Policy Iteration and Structured Value Iteration*

Structured Value Iteration (SVI) and *Structured Policy Iteration* (SPI) [BOU 00] are adaptations to FMDPs of the *Policy Iteration* and *Value Iteration* algorithms. In addition to using function-specific independence, SVI and SPI exploit context-specific independence by using decision trees to represent the different functions of the problem.

4.3.1.1. *Decision Trees*

Decision trees represent a function by partitioning its input space and associating the output value to each of these partitions. A decision tree is composed of:

– *internal nodes* (or decision nodes): they represent a test on a variable of the input space. They are parents of other nodes in the tree and define the partitions of the input space;

– *edges*: they connect a parent interior node to a child node and constrain the value of the variable tested at the parent node to one value to reach the child node;

– *external nodes* (or leaves): they represent the terminal nodes of the tree and define the value of the function for the partition defined by the parent (internal) nodes.

Note that, in a decision tree, one node has only one parent (except for the root which has no parent).

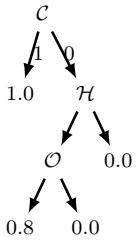
In SVI and SPI, decision trees are used to represent the different functions of the FMDP, such as reward functions, transition functions, policies and value functions. A function f represented by a decision tree is noted $\mathsf{Tree}\,[f]$. Graphically, we represent decision trees with the following convention: for an internal node testing a Boolean variable X, the left and right edges correspond respectively to $X = 1$ and $X = 0$ (or respectively X being true and X being false).

4.3.1.2. *Representation of the Transition Function*

In the *Coffee Robot* problem, the tabular representation of the conditional probability distribution $P_{\mathcal{D}\mathrm{elC}}(\mathcal{C}'|\mathcal{O},\mathcal{H},\mathcal{C})$ (see Figure 4.3(a)) exhibits, depending on the context, different regularities that can be exploited to represent the function more compactly. For instance, as described above in Section 4.2.4, in the context where $\mathcal{C} = 1$, the probability that $\mathcal{C}'$ is true is equal to 1, whatever the value of the two other variables $\mathcal{O}, \mathcal{H} \in \textsc{Parents}_{\mathcal{D}\mathrm{elC}}(\mathcal{C}')$. In the problem, this means that it is certain that an owner having a coffee will still have it at the next time step. Decision trees allow to represent such context-specific regularities more compactly than tabular representations.

$\mathcal{C}$	$\mathcal{H}$	$\mathcal{O}$	$\mathcal{C}'$
1	1	1	1.0
1	1	0	1.0
1	0	1	1.0
1	0	0	1.0
0	1	1	0.8
0	1	0	0.0
0	0	1	0.0
0	0	0	0.0

(a)



(b)

Figure 4.3. *Representation of the conditional probability distribution $P_{\mathcal{D}\mathrm{elC}}(\mathcal{C}'|\mathcal{O},\mathcal{H},\mathcal{C})$ with a tabular data structure (Figure a) and a decision tree (Figure b). The leaf noted 0.8 means that the probability for the variable $\mathcal{C}'$ of being true at the next time step is: $P_{\mathcal{D}\mathrm{elC}}(\mathcal{C}' = 1|\mathcal{O} = 1, \mathcal{H} = 1, \mathcal{C} = 0) = 0.8$. In the decision tree, note that some regularities are aggregated such the probability distributions when $P_{\mathcal{D}\mathrm{elC}}(\mathcal{C}' = 1|\mathcal{C} = 1) = 1.0$.*

A decision tree $\mathsf{Tree}\,[P_\tau(X'|\textsc{Parents}_\tau(X'))]$ representing a conditional probability distribution $P_\tau(X'|\textsc{Parents}_\tau(X'))$ is made of:

– *internal nodes*: they represent a test on a variable $X_j \in \textsc{Parents}_\tau(X')$;

– *edges*: they represent a value $x_j \in \text{DOM}(X_j)$ of a variable X_j tested at the parent node and defining a partition represented by the child node connected to the edge;

– *external nodes*: they represent the probability distribution $P_{\tau_l}(X'|c_l)$, with c_l the context defined by the set of values of the variables $X_j \in \text{PARENTS}_{\tau_l}(X')$ tested in the parents node for a leaf l in the tree.

Reading such a tree is straightforward: the probability distribution of a variable X' for a given instantiation x is given by the unique leaf reached by selecting the edge at each internal node corresponding to the value of the tested variable in the instantiation x. Such a path defines the context C_l associated to the leaf l reached.

Figure 4.3(b) represents the conditional probability distribution $P_{\mathcal{D}\text{elC}}(\mathcal{C}'|\mathcal{O},\mathcal{H},\mathcal{C})$ as a decision tree. The value at a leaf indicates the probability that the variable $\mathcal{C}'$ will be true at the next time step. Because the decision tree representation exploits context-specific independence in this example, the representation of $P_{\mathcal{D}\text{elC}}(\mathcal{C}'|\mathcal{O},\mathcal{H},\mathcal{C})$ is more compact compared to the tabular representation. Whereas 8 lines are required (Figure 4.3(a)) for the tabular form, only 4 are required for the same function with a decision tree. Such factorization is one of the main principle used by SPI and SVI for planning.

4.3.1.3. *Representation of the Reward Function*

The representation of the reward function with decision trees is very similar to the representation of conditional probability distributions described above. Indeed, the semantics of the internal nodes and edges are the same. Only the values attached to the leaves of the tree are different: rather than probabilities, the leaves represent real numbers.

Figure 4.4 represents the reward function for the *Coffee Robot* problem and compares the tabular representation $R(\boldsymbol{x})$ (Figure a) with a decision tree representation $\mathsf{Tree}\,[R(\boldsymbol{x})]$ (Figure b). Note that the number of leaves in the tree is equal to the number of lines in the table, meaning that there is no context-specific independence to exploit in the representation of this function.

Finally, SVI and SPI are not able to exploit the additive decomposition of the reward function as described in Section 4.2.3.4.

4.3.1.4. *Representation of a Policy*

Of course, a policy $\pi(\boldsymbol{x})$ can also be represented with a decision tree $\mathsf{Tree}\,[\pi(\boldsymbol{x})]$. Figure 4.5 represents a stationary optimal policy $\mathsf{Tree}\,[\pi^*(\boldsymbol{x})]$ in the *Coffee Robot* problem.

The state space of the problem *Coffee Robot* is composed of 6 binary variables. So, a tabular representation of a policy π requires $2^6 = 64$ entries. The tree $\mathsf{Tree}\,[\pi^*]$

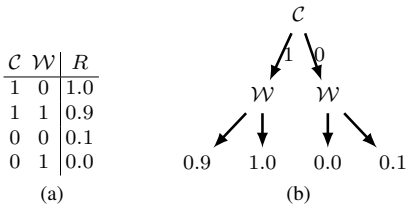


Figure 4.4. *Definition of the reward function $R(\boldsymbol{x})$ with a tabular representation (Figure a) and a decision tree (Figure b). The leaf noted 0.9 means $R(\mathcal{C}=1,\mathcal{W}=1)=0.9$.*

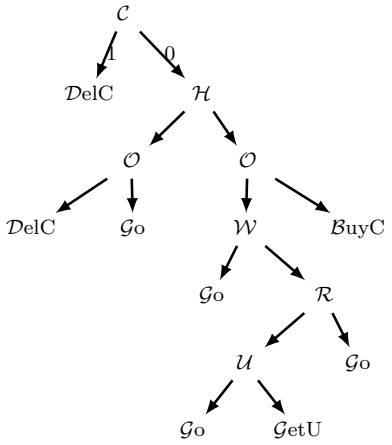


Figure 4.5. *Representation of an optimal policy $\pi^*(\boldsymbol{x})$ with a decision tree $\mathsf{Tree}\left[\pi^*(\boldsymbol{x})\right]$. The leaf noted $\mathcal{B}\mathrm{uyC}$ means $\pi(\mathcal{C}=0,\mathcal{H}=0,\mathcal{O}=0)=\mathcal{B}\mathrm{uyC}$.*

representing the optimal policy in the *Coffee Robot* problem requires only 8 leaves (15 nodes in total). Consequently, in this problem, the decision tree representation of the policy exploits context-specific independencies. This means, for instance, that when the robot is in the office with a coffee, it is not necessary to check the weather to define the best action to perform.

In the worth case, note that only N tests are required to determine the action to execute for a problem with a state space made of N variables. This is not necessarily the case for all structured representations (see for instance Section 4.3.3.4). Moreover, decision trees allow to compute the values of the minimum set of variables needed to define the next action to execute. Such property can be important when the policy is run in an environment where computing the value of a variable has a cost (computation time for instance).

4.3.1.5. *Representation of the Value Function*

Obviously, the value function V_π of a policy π can also be represented with a decision tree $\mathsf{Tree}\,[V_\pi]$. The semantics of such tree is identical to a tree representing the reward function: internal nodes, edges and leaves represent respectively a test on a variable, a value of the tested variable at the parent internal node and the value of the function in the corresponding partition. Figure 4.6 represents the value function of the policy $\mathsf{Tree}\,[\pi^*]$ represented in Figure 4.5.

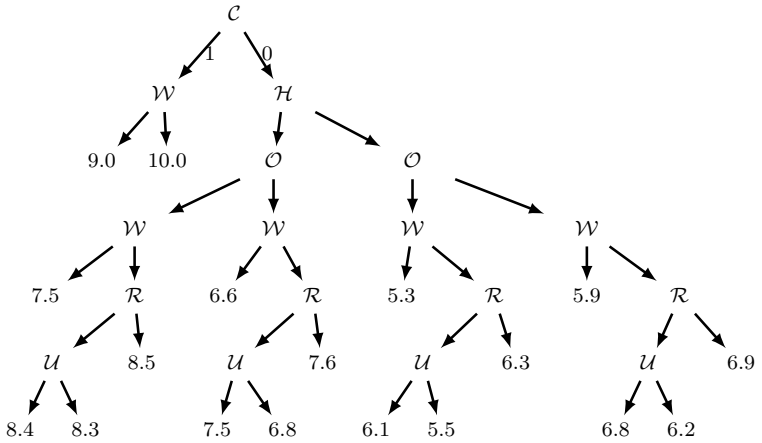


Figure 4.6. *Representation of the value function $V_{\pi^*}(\boldsymbol{x})$ of the policy π^* as a decision tree* $\mathsf{Tree}\,[V_{\pi^*}(\boldsymbol{x})]$ *for the problem* Coffee Robot. *The leaf noted* 10.0 *means* $V_{\pi^*}(\mathcal{C}=1,\mathcal{W}=0)=10.0$.

$\mathsf{Tree}\,[V_{\pi^*}(\boldsymbol{x})]$ contains only 18 leaves (35 nodes in total) whereas a tabular representation would have required 64 entries. Thus, in the *Coffee Robot* problem, a decision tree representation allows to exploit context-specific independencies. For instance, the value $V_{\pi^*}(\mathcal{C}=1,\mathcal{W}=0)$ of the optimal policy π^*, that is the owner has a coffee and the robot is dry, does not depend on the other variables in the problem. Consequently, the representation aggregates a set of states. Thus, when the value function is updated incrementally while solving the problem, it is required to compute only the value at the leaf corresponding to the context rather than updating every state corresponding to this same context.

However, a decision tree representation does not allow to exploit certain regularities in the structure of the function. For instance, the sub-trees in $\mathsf{Tree}\,[V_{\pi^*}]$ composed of the variables $\mathcal{R}$, $\mathcal{W}$, $\mathcal{U}$ and $\mathcal{O}$ share the same structure. Such structure can be exploited with an additive approximation of the value function as we will see in Section 4.3.3.5.

Finally, in the worst case, that is when the value function of the evaluated policy has a different value for each possible state, the size of the tree increases exponentially with the number of variables composing the state space, similarly to a tabular representation.

4.3.1.6. *Algorithms*

SVI and SPI are adaptations of, respectively, *Value Iteration* and *Policy Iteration* to decision tree representation. Consequently, rather than iterating on all the states of the problem to update the value function as *Value Iteration* and *Policy Iteration* do, SVI and SPI compute the update only for each leaf of the decision tree, decreasing the computation when states are aggregated and represented with one leaf. We recommend reading [BOU 00] for an exhaustive description of both SVI and SPI.

4.3.2. *SPUDD:* Stochastic Planning Using Decision Diagrams

In some problems, value functions have symmetries that are not exploited by decision trees, for instance when the function is strictly identical in some disjoint context. SPUDD (for *Stochastic Planning Using Decision Diagrams*), proposed by [HOE 99], uses *Algebraic Decision Diagrams* [BAH 93] (noted ADD) to represent the different functions of an FMDP. Similarly to SVI and SPI, SPUDD exploits function-specific and context-specific independencies.

Using ADDs rather than decision trees has two additional advantages. First of all, as mentioned before, ADDs can aggregate together identical substructures which have disjoint contexts.

Second, the variables used in an ADD are ordered. Whereas finding an optimal order of tests on the variables of the problem to represent the most compact representation is a difficult problem, [HOE 00] describes different heuristics that are good enough to improve significantly the size of the representation. Moreover, such an ordering is exploited to manipulate ADDs more efficiently compared to decision trees where no ordering is assumed.

Whereas SPUDD, similarly to SVI, is an adaptation of *Value Iteration* to work with ADDs, both of the advantages described above allow SPUDD to perform significantly better than SPI or SVI on most problems proposed in the FMDP literature.

4.3.2.1. *Representing the Functions of an FMDP with ADDs*

ADDs are a generalization of binary decision diagrams [BRY 86]. Binary decision diagrams are a compact representation of $\mathcal{B}^n \rightarrow \mathcal{B}$ functions of n binary variables to a binary variable. ADDs generalize binary decision diagrams to represent $\mathcal{B}^n \rightarrow \mathbb{R}$ functions of n binary variables to a real value in $\mathbb{R}$. An ADD is defined by:

– *internal nodes* (or decision nodes): they represent a test on a variable from the input space. They are the parent of two edges corresponding respectively to the values `true` and `false`;

– *edges*: they connect each parent internal node to a child node depending of its associated value `true` or `false`;

– *external nodes* (or leaves): they represent terminal nodes in the diagram and are associated to the value of the function in the subspace defined by the set of tests of the parent nodes to reach the leaf.

Unlike decision trees, a node (internal or external) in an ADD can have multiple parents. A function f represented with an ADD is noted $\mathsf{ADD}\,[f]$. We use the following graphical convention to represent ADDs: the edges of an internal node testing a variable X are drawn with a plain or dashed line, corresponding respectively to X being `true` or `false` (or $X = 1$ and $X = 0$).

Compared to decision trees, ADDs have several interesting properties. First, because an order is given, each distinct function has only one representation. Moreover, the size of the representation can be compacted because identical sub-graphs can be factored in the description. Finally, optimized algorithms have been proposed for most of the basic operators, such as the multiplication, the addition or the maximization of two ADDs.

Figure 4.7 shows an example of the same function f represented with a decision tree and an ADD. The figure illustrates that decision trees, unlike ADDs, are not adapted to represent disjunctive functions. Thus, the tree representation $\mathsf{Tree}\,[f]$ is composed of 5 different leaves (and 4 internal nodes) whereas the ADD representation $\mathsf{ADD}\,[f]$ contains only 2 leaves (and 3 internal nodes). Thus, an algorithm iterating on the leaves of the representation may have its complexity decreased when using ADDs rather than decision trees.

However, using ADDs adds two constraints on the FMDP to solve. First, all the variables in the FMDP have to be binary, ADDs representing only functions $\mathcal{B}^n \rightarrow \mathbb{R}$. For FMDPs with non binary variables, these variables are decomposed and replaced by their corresponding additional (binary) variables. Secondly, as mentioned above, the algorithms manipulating ADDs assume that, in the ADDs, the tests on the variables (the internal nodes) are sorted. When both constraints are satisfied, it is possible to represent all the functions of an FMDP with ADDs.

4.3.2.2. *Algorithm*

Similarly to SVI, SPUDD is based on *Value Iteration* with the operators implemented to manipulate ADDs, assuming that all the variables are binary and that they

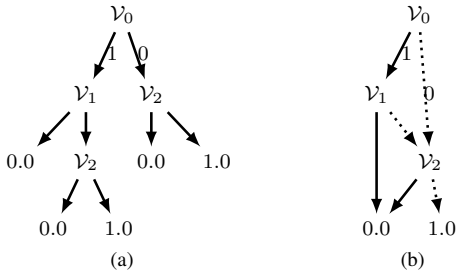


Figure 4.7. *Comparison of the representation of a function f as a decision tree* Tree $[f]$ *(Figure a) and as an algebraic decision diagram* ADD $[f]$ *(Figure b).*

are ordered. The work on SPUDD has led to APRICODD [STA 01] which contains additional improvements. First of all, the user can parameterize the algorithm to approximate the value function by limiting the maximum size of the ADD used to represent the value function. Moreover, APRICODD implements different methods for automatic variable ordering to avoid the user to have to specify it manually.

The last version of APRICODD is available on the Internet.[3] Note that APRICODD can be considered as a ready-to-use solution method to solve large problems that can be modeled as FMDPs.

4.3.3. *Approximate Linear Programming in FMDPs*

An alternative to dynamic programming to solve an MDP is linear programming (see Section 1.6.2.1). Using linear programming to solve FMDPs is the result of a work started by [KOL 99, KOL 00] and then continued with Guestrin [GUE 01, GUE 03a, GUE 03b].

The optimal value function of an MDP can be computed by formulating the MDP as a linear program [MAN 60]:

$$\begin{array}{ll} \text{For the variables:} & V(s), \forall s \in S\ ; \\ \text{Minimize:} & \sum_s \alpha(s) V(s)\ ; \\ \text{Under constraints:} & V(s) \geq R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \\ & \forall s \in S, \forall a \in A. \end{array} \qquad \text{(LP 1)}$$

where $\alpha(s) > 0$ is the state relevance weight for the state s.

3. http://www.cs.toronto.edu/~jhoey/spudd

Unfortunately, solving such a linear program is not possible for large MDPs because of the complexity of the objective function, of the number of variables to solve and of the number of constraints. These problems are solved by, first, using a linear approximation of the value function and, second, exploiting function-specific independence and additive decomposition of the reward function.

More precisely, using a linear approximation of the value function (that is a linear combination of basis functions [SCH 85]) decreases the complexity of the objective function to optimize and the number of variables to determine. Function-specific independence and additive decomposition of the reward function are exploited by an algorithm decomposing the constraints of the original linear program into a set of constraints with a complexity depending on the structure of the problem rather than on its size.

Both of these ideas are exploited by two different algorithms proposed in [GUE 03b]. The first one is based on the *Policy Iteration* algorithm using linear programming to evaluate the current policy. The second one constructs a linear program similar to (LP 1) to directly evaluate the optimal value function of the FMDP to solve. The next section presents the representation used by both algorithms.

4.3.3.1. *Representations*

Two different representations are used by the algorithms proposed by [GUE 03b]. The first representation is the tabular representation (similar to the tabular representation used in Figure 4.1, Section 4.2.3.3). Algorithms using such representation exploit function-specific independencies, linear approximation of the value function and additive decomposition of the reward function (and not context-specific independencies).

The second representation is a structured representation based on rules [ZHA 99], allowing to exploit context-specific independencies in a function. Whereas [GUE 03b] shows that, for some problems, tabular representations are faster, we have chosen to describe the rules representation mainly because the complexity of the worst case using these representations is better than the worst case of tabular representations [STA 01, GUE 03a].

[GUE 03b] prefers using rules rather than another structured representation because rules may not be exclusive, unlike decision trees or ADDs. We distinguish two types of rules: *probability rules* and *value rules*. The former are used to represent the transition function, the latter to represent value and reward functions. We describe how these rules are used to represent the functions of an FMDP in the following sections. A function f is noted $\mathsf{Rule}[f]$ when represented with a set of rules.

4.3.3.2. *Representation of the Transition Function*

Probability rules describe the transition function in an FMDP. More precisely, they are used to define the conditional probability distributions quantifying the DBNs. A rule corresponds to one context defining the same probability for this context.

We first start by defining the consistency between two contexts:

DEFINITION 4.3.– *Consistency between two contexts*
Let $\boldsymbol{C} \subseteq \{\boldsymbol{X}, \boldsymbol{X}'\}$, $\boldsymbol{c} \in \text{DOM}(\boldsymbol{C})$, $\boldsymbol{B} \subseteq \{\boldsymbol{X}, \boldsymbol{X}'\}$ *and* $\boldsymbol{b} \in \text{DOM}(\boldsymbol{B})$. *Two contexts* $\boldsymbol{b}$ *and* $\boldsymbol{c}$ *are consistent if they have the same assignment for the variables in the intersection* $\boldsymbol{C} \cap \boldsymbol{B}$.

Consequently, identical probabilities with consistent contexts are represented with probability rules:

DEFINITION 4.4.– *Probability rule*
A probability rule $\eta = |\boldsymbol{c} : p|$ *is a function* $\eta : \{\boldsymbol{X}, \boldsymbol{X}'\} \rightarrow [0, 1]$ *with the context* $\boldsymbol{c} \in \text{DOM}(\boldsymbol{C})$, $\boldsymbol{C} \subseteq \{\boldsymbol{X}, \boldsymbol{X}'\}$ *and* $p \in [0, 1]$, *and such that* $\eta(\boldsymbol{x}, \boldsymbol{x}') = p$ *if the instantiations* $\boldsymbol{x}$ *and* $\boldsymbol{x}'$ *are consistent with* $\boldsymbol{c}$, *or else equal to 1.*

Two rules are consistent if their context is consistent.

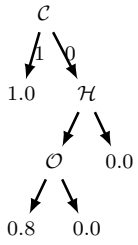
A set of probability rules completely defines a conditional probability distribution:

DEFINITION 4.5.– *Set of probability rules*
A set of rules P_a *of a conditional probability distribution is a function* $P_a : (\{X'_i\} \cup \boldsymbol{X}) \rightarrow [0, 1]$ *composed of the probability rules* $\{\eta_1, \ldots, \eta_m\}$ *with their mutually exclusive and exhaustive contexts. We define:* $P_a(x'_i|\boldsymbol{x}) = \eta_j(\boldsymbol{x}, \boldsymbol{x}')$ *with* η_j *the only rule in* P_a *with the context* $\boldsymbol{c}_j$ *consistent with* $(x'_i, \boldsymbol{x})$. *Moreover, we necessarily have:* $\forall \boldsymbol{x} \in \boldsymbol{X} : \sum_{x'_i} P_a(x'_i|\boldsymbol{x}) = 1$.

Note that $\text{PARENTS}_a(X'_i)$ can be defined as the union of the variables appearing in the contexts of the rules defining the distribution.

Similarly to decision trees, the sets of rules allow to exploit context-specific independencies. Moreover, decision trees define a complete partition of a space. Thus, it is straightforward to define a set of mutually exclusive and exhaustive rules from a given decision tree, as shown in Figure 4.8 for the conditional probability distribution $P_{\mathcal{D}\text{elC}}(\mathcal{C}'|\text{PARENTS}_{\mathcal{D}\text{elC}}(\mathcal{C}'))$.

The probability $P_{\mathcal{D}\text{elC}}(\mathcal{C}' = 1|\mathcal{C} = 0, \mathcal{O} = 1, \mathcal{H} = 1) = 0.8$ is represented by the corresponding rule $|\mathcal{C} = 0 \wedge \mathcal{O} = 1 \wedge \mathcal{H} = 1 \wedge \mathcal{C}' = 1 : 0.8|$. One can notice that the context of a rule is split in two parts. The first part is the set of tests on the variables X at time t, corresponding to the path in the tree to reach the leaf 0.8. The second part is



(a)

$$\left|\begin{array}{ll} \mathcal{C}=1 \wedge \mathcal{C}'=1 & : 1.0 \\ \mathcal{C}=0 \wedge \mathcal{O}=0 \wedge \mathcal{C}'=0 & : 1.0 \\ \mathcal{C}=0 \wedge \mathcal{O}=1 \wedge \mathcal{H}=1 \wedge \mathcal{C}'=0 & : 0.2 \\ \mathcal{C}=0 \wedge \mathcal{O}=1 \wedge \mathcal{H}=1 \wedge \mathcal{C}'=1 & : 0.8 \\ \mathcal{C}=0 \wedge \mathcal{O}=1 \wedge \mathcal{H}=0 \wedge \mathcal{C}'=0 & : 1.0 \end{array}\right|$$

(b)

Figure 4.8. *Representation of the conditional probability distribution* $P_{\mathcal{D}\text{elC}}(\mathcal{C}'|\text{PARENTS}_{\mathcal{D}\text{elC}}(\mathcal{C}'))$ *as a decision tree and a set of rules. The rule* $|\mathcal{C}=0 \wedge \mathcal{O}=1 \wedge \mathcal{H}=1 \wedge \mathcal{C}'=1 : 0.8|$ *defines* $P_{\mathcal{D}\text{elC}}(\mathcal{C}'=1|\mathcal{C}=0,\mathcal{O}=1,\mathcal{H}=1)=0.8$.

the value of the variable X_i' at time $t+1$. Such representation is advantageous to solve problems with synchronous arcs. A conditional probability distribution f represented with a set of probability rules is noted $\mathsf{Rule_p}\,[f]$.

4.3.3.3. *Representation of the Reward Function*

We define value rules to represent the reward function of an FMDP:

DEFINITION 4.6.– *Value rule*
A value rule $\rho = |\boldsymbol{c} : v|$ *is a function* $\rho : \boldsymbol{X} \rightarrow \mathbb{R}$ *such that* $\rho(\boldsymbol{x}) = v$ *when* $\boldsymbol{x}$ *is consistent with the context* $\boldsymbol{c}$ *and else 0.*

Note that the scope of a value rule may be defined as $\text{SCOPE}(\rho) = \boldsymbol{C}$ with $\boldsymbol{C}$ the set of instantiated variables in the context $\boldsymbol{c}$ of the rule $\rho = |\boldsymbol{c} : v|$.

It is now possible to define a function as a set of value rules:

DEFINITION 4.7.– *Set of a value rule*
A set of value rules representing a function $f : \boldsymbol{X} \rightarrow \mathbb{R}$ *is composed of the set of value rules* $\{\rho_1, \ldots, \rho_n\}$ *such that* $f(\boldsymbol{x}) = \sum_{i=1}^{n} \rho_i(\boldsymbol{x})$ *with* $\forall i : \text{SCOPE}(\rho_i) \subseteq \boldsymbol{X}$.

A function f represented with a set of value rules is noted $\mathsf{Rule_v}\,[f]$.

Moreover, [GUE 03b] assumes that a reward function $R(\boldsymbol{x}, a)$ can be specified as the sum of reward functions with a limited scope:

$$R(\boldsymbol{x}, a) = \sum_j r_j^a(\boldsymbol{x}). \tag{4.4}$$

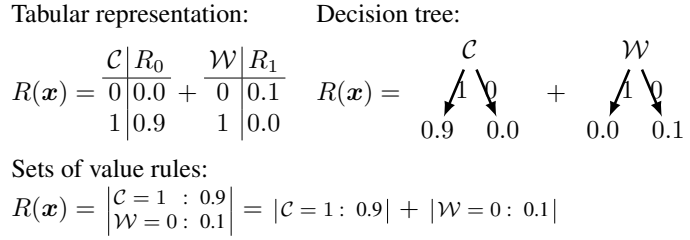


Figure 4.9. *Representation of the reward function R in the* Coffee Robot *problem. The reward is decomposed as a sum of reward functions with a scope limited to only one variable for each function.*

As shown in Figure 4.9, such representation allows to easily define functions exploiting at the same time context-specific independence and additive decomposition.

As described in Section 4.2.3.4, the reward function in the *Coffee Robot* problem can be decomposed as a sum of two reward functions, each with a scope limited to only one variable of the problem. Different representations can be used to define these functions, in particular tables, decision trees or sets of rules. [GUE 03b] uses sets of value rules.

4.3.3.4. *Policy Representation*

To compactly represent a policy π, [GUE 03b] uses a data structure first proposed by [KOL 00]. Rather than using $\mathsf{Tree}[\pi]$ or $\mathsf{ADD}[\pi]$, a default action in the FMDP is defined a priori and a policy is represented by an ordered decision list.

Every element in the list is defined by a triple containing: a context defining whether the action can be executed for a given state s, the action to execute if this decision has been taken and a bonus corresponding to the additional expected long term reward compared to the expected long term reward if the default action were taken. The last element of a policy is always the default action with an empty context (the decision that can be taken at any time) and a bonus of 0. A policy π represented as a decision list is noted $\mathsf{List}[\pi]$. Table 4.1 shows the optimal policy for the *Coffee Robot* problem where the default action is $\mathcal{G}\mathrm{o}$.

Note that, unlike decision trees or ADDs, the number of tests required to determine the action to execute can be superior to the number of variables in the problem.

4.3.3.5. *Representation of the Value Function*

We have seen that an MDP can be specified as the following linear program (Section 1.6.2.1, LP 1):

	Context	Action	Bonus
0	$\mathcal{C}=0 \wedge \mathcal{H}=1 \wedge \mathcal{W}=0 \wedge \mathcal{R}=1 \wedge \mathcal{U}=0 \wedge \mathcal{O}=1$	$\mathcal{D}$elC	2.28
1	$\mathcal{C}=0 \wedge \mathcal{H}=0 \wedge \mathcal{W}=0 \wedge \mathcal{R}=1 \wedge \mathcal{U}=0 \wedge \mathcal{O}=0$	$\mathcal{B}$uyC	1.87
2	$\mathcal{C}=0 \wedge \mathcal{H}=1 \wedge \mathcal{W}=0 \wedge \mathcal{R}=1 \wedge \mathcal{U}=1 \wedge \mathcal{O}=1$	$\mathcal{D}$elC	1.60
3	$\mathcal{C}=0 \wedge \mathcal{H}=1 \wedge \mathcal{W}=1 \wedge \mathcal{O}=1$	$\mathcal{D}$elC	1.45
4	$\mathcal{C}=0 \wedge \mathcal{H}=1 \wedge \mathcal{W}=0 \wedge \mathcal{R}=0 \wedge \mathcal{O}=1$	$\mathcal{D}$elC	1.44
5	$\mathcal{C}=0 \wedge \mathcal{H}=0 \wedge \mathcal{W}=0 \wedge \mathcal{R}=1 \wedge \mathcal{U}=1 \wedge \mathcal{O}=0$	$\mathcal{B}$uyC	1.27
6	$\mathcal{C}=0 \wedge \mathcal{H}=0 \wedge \mathcal{W}=1 \wedge \mathcal{O}=0$	$\mathcal{B}$uyC	1.18
7	$\mathcal{C}=0 \wedge \mathcal{H}=0 \wedge \mathcal{W}=0 \wedge \mathcal{R}=0 \wedge \mathcal{O}=0$	$\mathcal{B}$uyC	1.18
8	$\mathcal{C}=1 \wedge \mathcal{W}=0 \wedge \mathcal{R}=1 \wedge \mathcal{U}=0$	$\mathcal{D}$elC	0.84
9	$\mathcal{C}=0 \wedge \mathcal{H}=0 \wedge \mathcal{W}=0 \wedge \mathcal{R}=1 \wedge \mathcal{U}=0 \wedge \mathcal{O}=1$	$\mathcal{G}$etU	0.18
10	$\mathcal{C}=1 \wedge \mathcal{W}=0 \wedge \mathcal{R}=1 \wedge \mathcal{U}=1$	$\mathcal{D}$elC	0.09
11	$\emptyset$	$\mathcal{G}$o	0.00

Table 4.1. *Representation of policy $\pi(s)$ as a decision list* List $[\pi]$ *(with the default action $\mathcal{G}$o).*

$$\begin{array}{ll} \text{For the variables:} & V(s), \forall s \in S\ ; \\ \text{Minimize:} & \sum_s \alpha(s) V(s)\ ; \\ \text{Under constraints:} & V(s) \geq R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \\ & \forall s \in S, \forall a \in A. \end{array} \quad \text{(LP 2)}$$

However, as described in Section 4.3.3, because of the complexity in the number of variables to determine, the number of terms in the sum of the objective function and the number of constraints, it is not possible to solve this linear program for large problems.

One solution to decrease the number of terms in the sum of the objective function and the number of variables to solve is to approximate the value function with a *linear combination* proposed by [BEL 63] (see Chapter 3). The space of approximate value functions $\tilde{V} \in \mathcal{H} \subseteq \mathbb{R}^n$ is defined by a set of *basis functions* with a scope limited to a small number of variables:

DEFINITION 4.8.– *Linear value function*
A linear value function $\tilde{V}$ with a set $H = \{h_0, \ldots, h_k\}$ of basis functions is a function such that $\tilde{V}(s) = \sum_{j=1}^{k} w_j h_j(s)$ with $w \in \mathbb{R}^k$.

Such an approximation can directly be used to redefine the linear program by simply replacing the value function by its approximation in the objective function of the linear program [SCH 85]:

$$\begin{array}{ll} \text{For the variables:} & w_1, \ldots, w_k\ ; \\ \text{Minimize:} & \sum_s \alpha(s) \sum_{i=1}^{k} w_i h_i(s)\ ; \\ \text{Under constraints:} & \sum_{i=1}^{k} w_i h_i(s) \geq R(s,a) + \\ & \qquad \gamma \sum_{s'} P(s'|s,a) \sum_{i=1}^{k} w_i h_i(s') \\ & \forall s \in S, \forall a \in A. \end{array} \quad \text{(LP 3)}$$

Consequently, rather than determining the value function in the complete space of value functions, the search is reduced to the space corresponding to the set of weights w_i used in the linear approximation. Moreover, limiting the scope of the basis functions allows to exploit function-specific independence to reduce the number of constraints.

However, whereas the number of variables to determine is not the number of possible states in the problem anymore but the number of weights w_i in the approximation, the number of terms in the sum and the number of constraints are still equal to the number of states in the problem.

For such linear program, a solution exists only if a constant basis function is included in the set of basis functions [SCH 85]. [GUE 03b] assumes that such a function h_0, such that $h_0(s) = 1, \forall s \in S$, is systematically included in the set of basis functions. Additionally, unlike (LP 1), the state relevance weights $\alpha(s)$ have an important effect on the quality of the approximation [FAR 01] and, thus, on the quality of the policies computed from the value function.

Such an approximation in the value function allows to exploit at the same time function-specific independencies and to exploit additional regularities in the structure of the value function as shown for the *Coffee Robot* problem in Figure 4.10. The additive decomposition of the approximated value function allows to exploit regularities that neither decision trees nor ADDs are able to exploit, such as the similarities in the structure of internal nodes.

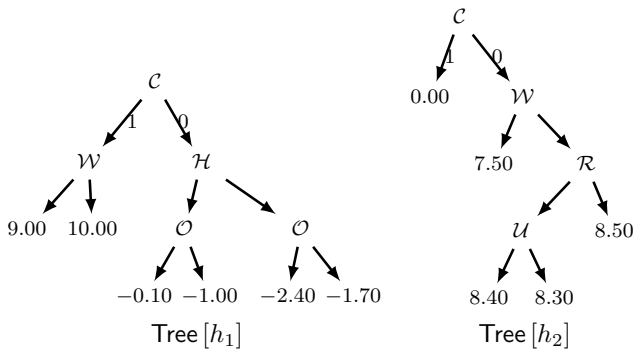


Figure 4.10. *Example of a linear combination of a value function in the* Coffee Robot *problem as two decision trees representing two basis functions (with limited scopes) corresponding to the optimal policy $\pi^*(s)$ (Table 4.1). The optimal approximated value function is: $\tilde{V}^*(s) = 0.63 \cdot \mathsf{Tree}[h_0] + 0.94 \cdot \mathsf{Tree}[h_1] + 0.96 \cdot \mathsf{Tree}[h_2]$. The tree $\mathsf{Tree}[h_0]$ is not shown since it defines a constant basis function and contains only one leaf equal to 1.*

The definition of the value function $\mathsf{Tree}[V]$ (see Figure 4.6 in Section 4.3.1) is decomposed in two basis functions $\mathsf{Tree}[h_1]$ and $\mathsf{Tree}[h_2]$ and allows an approximation of $\mathsf{Tree}[V_{\pi^*}]$ with an error inferior to 1.0.[4] Additive decomposition of the value function is exploited because rather than containing 18 leaves, this representation requires only 11 leaves for both trees. So, the approximation of the value function in the *Coffee Robot* problem is composed of three basis functions (including the constant basis function). Thus, three weights, w_0, w_1 and w_2, must be determined with (LP 2).

Finally, when the reward function is compactly represented using an additive decomposition, it seems natural to expect that the value function also exhibits this kind properties. However, this is not necessarily the case. Indeed, a problem with a reward function with no additive decomposition may have an optimal value function well approximated with a linear approximation. On the other hand, a compact representation of the reward function or the transition function does not imply a compact representation of the value function [KOL 99, MUN 00, LIB 02].

4.3.3.6. *Algorithms*

Consequently, the algorithms proposed by [GUE 03b], for an FMDP, generate a linear program to compute the value function of the problem. Additional algorithms are also described to compute a policy (as a decision list). However, such a representation can be very expensive, even intractable in some cases. So, the authors suggest to directly estimate the approximated optimal value function of the problem. Then, approximated action value functions are computed for each action (using the FMDP) to compare actions between each others and to estimate the best action to execute for a given state. Thus, an explicit representation of the policy is avoided. We suggest to refer to [GUE 03b] and [GUE 03a] for a complete description of these algorithms.

4.4. Perspectives and Conclusion

Solving large FMDPs is still an active field of research and different extensions have been proposed in the last ten years. One extension, studied in [POU 05], is the extension to partially observable problems. Other extensions have been proposed to avoid having to specify the full structure or the values of the FMDP.

In this context, the algorithms presented in Section 2.6.2, have been adapted to FMDPs, namely DBN-E^3 [KEA 99], *factored* R-MAX [STR 07] and *factored* I.E. [STR 07]. These algorithms assume that function-specific independencies are known but not quantified. These algorithms propose exploring strategies to reach a policy near an optimal policy of the FMDP in a finite time.

---

4. The basis functions $\mathsf{Tree}[h_1]$ and $\mathsf{Tree}[h_2]$ have been defined manually, knowing $\mathsf{Tree}[V_{\pi^*}]$ in the *Coffee Robot* problem.

A second approach does not assume that function-specific independencies are known beforehand and learn the structure of the problem from trials and errors of an agent in the problem [DEG 06]. However, despite interesting experimental results, no proof has been proposed yet. Research in this context is still being active [STR 07].

Another field of research in FMDPs is that of hierarchical approaches. A hierarchy of sub-problems is defined directly from a given FMDP. In this context, [JON 06] propose an algorithm named VISA with similar or better performance than SPUDD on different problems.

Finally, [SZI 08] have proposed to use dynamic programming rather than linear programming to solve FMDPs with a value function approximated by a linear combination of basis functions. Though their approach does not have necessarily better performance on all problems, their algorithms are notably simpler than the ones proposed by [GUE 03b].

4.5. Bibliography

[BAH 93] BAHAR R., FROHM E., GAONA C., HACHTEL G., MACII E., PARDO A., SOMENZI F., "Algebraic Decision Diagrams and their Applications", *Proceedings of the IEEE/ACM International Conference on CAD*, Santa Clara, California, p. 188–191, 1993.

[BEL 63] BELLMAN R., KALABA R., KOTKIN B., "Polynomial Approximation - a New Computational Technique in Dynamic Programming", *Math. Comp.*, vol. 17, num. 8, p. 155–161, 1963.

[BOU 95] BOUTILIER C., DEARDEN R., GOLDSZMIDT M., "Exploiting Structure in Policy Construction", *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, Montreal, p. 1104–1111, 1995.

[BOU 96] BOUTILIER C., GOLDSZMIDT M., "The Frame Problem and Bayesian Network Action Representations", *Proceedings of the 11th Biennial Canadian Conference on Artificial Intelligence (AI '96)*, Toronto, CA, p. 69–83, 1996.

[BOU 99] BOUTILIER C., DEAN T., HANKS S., "Decision-Theoretic Planning: Structural Assumptions and Computational Leverage", *Journal of Artificial Intelligence Research*, vol. 11, p. 1–94, 1999.

[BOU 00] BOUTILIER C., DEARDEN R., GOLDSZMIDT M., "Stochastic Dynamic Programming with Factored Representations", *Artificial Intelligence*, vol. 121, num. 1, p. 49–107, 2000.

[BRY 86] BRYANT R. E., "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Transactions on Computers*, vol. C-35, num. 8, p. 677–691, 1986.

[DEG 06] DEGRIS T., SIGAUD O., WUILLEMIN P.-H., "Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems", *Proceedings of the 23rd International Conference on Machine Learning (ICML'06)*, Pittsburgh, Pennsylvania, USA, p. 257–264, 2006.

[FAR 01] DE FARIAS D., VAN ROY B., "The Linear Programming Approach to Approximate Dynamic Programming", *Operations Research*, vol. 51, num. 6, p. 850–856, 2001.

[GUE 01] GUESTRIN C., KOLLER D., PARR R., "Max-norm Projections for Factored MDPs", *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, p. 673–680, 2001.

[GUE 03a] GUESTRIN C., Planning Under Uncertainty in Complex Structured Environments, PhD thesis, Computer Science Department, Stanford University, USA, 2003.

[GUE 03b] GUESTRIN C., KOLLER D., PARR R., VENKATARAMAN S., "Efficient Solution Algorithms for Factored MDPs", *Journal of Artificial Intelligence Research*, vol. 19, p. 399–468, 2003.

[HOE 99] HOEY J., ST-AUBIN R., HU A., BOUTILIER C., "SPUDD: Stochastic Planning using Decision Diagrams", *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI'99)*, San Mateo, CA, Morgan Kaufmann, p. 279–288, 1999.

[HOE 00] HOEY J., ST-AUBIN R., HU A., BOUTILIER C., Optimal and Approximate Stochastic Planning using Decision Diagrams, Report num. TR-00-05, University of British Columbia, 2000.

[JON 06] JONSSON A., BARTO A., "Causal Graph Based Decomposition of Factored MDPs", *Journal of Machine Learning Research*, vol. 7, p. 2259–2301, 2006.

[KEA 99] KEARNS M., KOLLER D., "Efficient Reinforcement Learning in Factored MDPs", *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, 1999.

[KOL 99] KOLLER D., PARR R., "Computing Factored Value Functions for Policies in Structured MDPs", *Proceedings 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, p. 1332–1339, 1999.

[KOL 00] KOLLER D., PARR R., "Policy Iteration for Factored MDPs", *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI'00)*, p. 326–334, 2000.

[LIB 02] LIBERATORE P., "The size of MDP factored policies", *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'02)*, p. 267–272, 2002.

[MAN 60] MANNE A. S., *Linear Programming and Sequential Decisions*, Cowles Foundation for Research in Economics at Yale University, 1960.

[MUN 00] MUNDHENK M., GOLDSMITH J., LUSENA C., ALLENDER E., "Complexity of Finite-Horizon Markov Decision Process Problems", *Journal of the ACM (JACM)*, vol. 47, num. 4, p. 681–720, ACM Press New York, NY, USA, 2000.

[PEA 88] PEARL J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Mateo, CA, 1988.

[POO 97] POOLE D., "The Independent Choice Logic for Modelling Multiple Agents under Uncertainty", *Artificial Intelligence*, vol. 94, num. 1-2, p. 7–56, 1997.

[POU 05] POUPART P., Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes, PhD thesis, University of Toronto, 2005.

[RIV 87] RIVEST R. L., "Learning Decision Lists", *Machine Learning*, vol. 2, p. 229–246, 1987.

[SCH 85] SCHWEITZER P., SEIDMANN A., "Generalized Polynomial Approximations in Markovian Decision Processes", *Journal of Mathematical Analysis and Applications*, vol. 110, p. 568–582, 1985.

[STA 01] ST-AUBIN R., HOEY J., BOUTILIER C., "APRICODD: Approximate Policy Construction Using Decision Diagrams", *Advances in Neural Information Processing Systems 13 (NIPS'00)*, p. 1089–1095, 2001.

[STR 07] STREHL A., DIUK C., LITTMAN M. L., "Efficient Structure Learning in Factored-state MDPs", *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI'07)*, 2007.

[SZI 08] SZITA I., LÖRINCZ A., "Factored value iteration converges", *Acta Cybernetica*, vol. 18, num. 4, p. 615–635, 2008.

[ZHA 99] ZHANG T., POOLE D., "On the Role of Context-specific Independence in Probabilistic Reasoning", *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, Stockholm, p. 1288–1293, 1999.