# Chapter 3
# Approximate Dynamic Programming by Practical Examples

Martijn R.K. Mes and Arturo Pérez Rivera

**Abstract** Computing the exact solution of an MDP model is generally difficult and possibly intractable for realistically sized problem instances. A powerful technique to solve the large scale discrete time multistage stochastic control processes is Approximate Dynamic Programming (ADP). Although ADP is used as an umbrella term for a broad spectrum of methods to approximate the optimal solution of MDPs, the common denominator is typically to combine optimization with simulation, use approximations of the optimal values of the Bellman's equations, and use approximate policies. This chapter aims to present and illustrate the basics of these steps by a number of practical and instructive examples. We use three examples (1) to explain the basics of ADP, relying on value iteration with an approximation of the value functions, (2) to provide insight into implementation issues, and (3) to provide test cases for the reader to validate its own ADP implementations.

**Key words:** Dynamic programming, Approximate dynamic programming, Stochastic optimization, Monte Carlo simulation, Curse of dimensionality

## 3.1 Introduction

*Approximate Dynamic Programming (ADP) is a powerful technique to solve large scale discrete time multistage stochastic control processes, i.e., complex Markov Decision Processes (MDPs).*

M.R.K. Mes (✉) • A. Pérez Rivera
Department of Industrial Engineering and Business Information Systems, University of Twente, Enschede, The Netherlands
e-mail: m.r.k.mes@utwente.nl

© Springer International Publishing AG 2017
R.J. Boucherie, N.M. van Dijk (eds.), *Markov Decision Processes in Practice*,
International Series in Operations Research & Management Science 248,
DOI 10.1007/978-3-319-47766-4_3

MDPs consist of a state space $\mathcal{S}$, and at each time step $t$, the system is in a particular state $S_t \in \mathcal{S}$ from which we can take a decision $x_t$ from the feasible set $\mathcal{X}_t$. This decision results in rewards or costs, typically given by $C_t(S_t, x_t)$, and brings us to a new state $S_{t+1}$ with probability $\mathbb{P}(S_{t+1}|S_t, x_t)$, i.e., the next state is conditionally independent of all previous states and actions. Therefore, the decision not only determines the direct costs, but also the environment within which future decisions take place, and hence influences the future costs. The goal is to find a policy. A policy $\pi \in \Pi$ can be seen as a decision function $X^{\pi}(S_t)$ that returns a decision $x_t \in \mathcal{X}_t$ for all states $S_t \in \mathcal{S}$, with $\Pi$ being the set of potential decision functions or policies. The problem of finding the best policy can be written as

$$\min_{\pi \in \Pi} \mathbb{E}^{\pi} \left\{ \sum_{t=0}^{T} \gamma C_t(S_t, X_t^{\pi}(S_t)) \right\}, \tag{3.1}$$

where $\gamma$ is a discount factor (minimizing the total and average rewards is achieved by setting $\gamma = 1$ and $\gamma = 1/T$ respectively), and $T$ denotes the planning horizon (could be infinite).

The problem formulation (3.1) covers a huge variety of decision problems, found in various applications also covered in this book, such as health care, production and manufacturing, communications, and transportation. There is also a wide variety of methods to solve (3.1), such as rolling-horizon procedures, simulation optimization, linear programming, and dynamic programming. Here, we focus on the latter.

Dynamic programming solves complex MDPs by breaking them into smaller subproblems. The optimal policy for the MDP is one that provides the optimal solution to all sub-problems of the MDP [1]. This becomes visible in Bellman's equation, which states that the optimal policy can be found by solving:

$$V_t(S_t) = \min_{x_t \in \mathcal{X}_t} \left( C_t(S_t, x_t) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(S_{t+1} = s'|S_t, x_t) V_{t+1}(s') \right). \tag{3.2}$$

Computing the exact solution, e.g., by using backward dynamic programming, is generally difficult and possibly intractable for large problems due to the widely known "curse of dimensionality". One can encounter three curses of dimensionality in dynamic programming [10]:

1. the state space $\mathcal{S}$ for the problem may be too large to evaluate the value function $V_t(S_t)$ for all states within reasonable time;
2. the decision space $\mathcal{X}_t$ may be too large to find the optimal decision for all states within reasonable time;
3. computing the expectation of 'future' costs may be intractable when the outcome space is large.

The outcome space is the set of possible states in time period $t+1$, given the state and decision in time period $t$. Its size is driven by the random information $W_{t+1}$ that arrives between $t$ and $t+1$. By capturing all the randomness with $W_{t+1}$, we can express the next state $S_{t+1}$ as a function of the current state $S_t$, the decision $x_t$, and the new information $W_{t+1}$, using a transition function $S_{t+1} = S^M(S_t, x_t, W_{t+1})$. Now, assume that $W_{t+1}$ is independent on all prior information, and let $\Omega_{t+1}$ be the set of possible outcomes of $W_{t+1}$ and let $\mathbb{P}(W_{t+1} = \omega)$ denote the probability of outcome $\omega \in \Omega_{t+1}$. We now rewrite (3.2) as

$$V_t(S_t) = \min_{x_t \in \mathcal{X}_t} \left( C_t(S_t, x_t) + \gamma \sum_{\omega \in \Omega_{t+1}} \mathbb{P}(W_{t+1} = \omega) V_{t+1}(S_{t+1} | S_t, x_t, \omega) \right), \quad (3.3)$$

with $S_{t+1} = S^M(S_t, x_t, \omega)$. Note that in (3.2) and (3.3) the direct costs $C_t(S_t, x_t)$ are assumed to be deterministic; however, the random information $\omega$ could also play a role in this function, see [10].

Based on an MDP model, Approximate Dynamic Programming (ADP) is a modeling framework that offers several strategies for tackling the curses of dimensionality in large, multi-period, stochastic optimization problems [10]. Also for ADP, the output is a policy or decision function $X_t^\pi(S_t)$ that maps each possible state $S_t$ to a decision $x_t$, for each stage $t$ in the planning horizon. Although ADP is used as an umbrella term for a broad spectrum of methods to approximate the optimal solution of MDPs, the common denominator is typically to combine optimization with simulation (sampling from $\Omega_{t+1}$), use approximations of the optimal values of the Bellman's equations, and use approximate policies. For applications of ADP, a more natural form of the Bellman's equations in (3.3) is the expectational form given by:

$$V_t(S_t) = \min_{x_t \in \mathcal{X}_t} \left( C_t(S_t, x_t) + \gamma \mathbb{E}^\omega \{ V_{t+1}(S_{t+1} | S_t, x_t, \omega) \} \right). \quad (3.4)$$

To approximate the optimal values of (3.4), a series of constructs and algorithmic manipulations of the MDP model are needed. This chapter aims to present and illustrate the basics of these steps by a number of practical and instructive examples.

ADP can be applied to large-scale instances because it is able to handle two of the three dimensionality issues. First, the large outcome space can be handled through the construct of a post-decision state $S_t^{x,n}$, which we explain in Sect. 3.2. Second, the large state space can be handled by (1) generating sample paths through the states, the so-called "forward dynamic programming" algorithmic strategy, which solves the Bellman's equations by stepping forward in time, and repeating this process for $N$ iterations, and (2) using approximate value functions $\overline{V}_t^n(S_t^{x,n})$ that are "learned" through the iterations and that might allow generalization across states, i.e., instead of learning the values of each state individually, visited states might tell us something about the value of states not visited yet. We also elaborate on this strategy in the next sections.

There are numerous ADP methods and variations available, each having their merits. Variations particularly consists in the type of value function approximations (e.g., lookup, parameterized, statistical) and policies used (e.g., policy approximations, tree search, roll-out heuristics, rolling horizon policies). In Sect. 3.5 we discuss some of these variations. But first, we explain the basic concept of ADP by means of three example problems. We use the examples (1) to explain the basics of ADP, relying on value iteration with an approximation of the value functions, (2) to provide insight into implementation issues, and (3) to provide test cases for the reader to validate its own ADP implementations (for the first two examples, we provide all experimental settings, ADP results, as well as the exact results). For each of the three examples we use a similar division, consisting of problem introduction, the MDP model with exact results, and the ADP approach. We introduce the basics of ADP using a first transportation example in Sect. 3.2. Next, we present ADP extensions using another transportation example in Sect. 3.3 and using an example application of ADP in healthcare in Sect. 3.4. As an aid to the reader, the notation used throughout these examples is summarized in Table 3.1.

| Variable | Description |
|---|---|
| $S \in \mathcal{S}$ | State |
| $x \in \mathcal{X}$ | Decision |
| $\pi \in \Pi$ | Policy |
| $X^{\pi}(S_t)$ | Decision function returning a decision in state $S_t$ under $\pi$ |
| $C_t(S_t, x_t)$ | Costs function giving the costs of decision $x_t$ in state $S_t$ |
| $\gamma$ | Discount factor |
| $T$ | Length planning horizon |
| $\mathbb{P}(S_{t+1} \| S_t, x_t)$ | Transition probability to $S_{t+1}$ given $S_t$ and $x_t$ |
| $W_{t+1} \in \Omega_{t+1}$ | Random information that arrives between $t$ and $t+1$ |
| $V_t(S_t)$ | Value function |
| $S^m(S_t, x_t, W_{t+1})$ | Transition function to $S_{t+1}$ given $S_t$, $x_t$, and $W_{t+1}$ |
| $S_t^x$ | Post-decision state |
| $S^{M,x}(S_t, x_t)$ | Transition function to $S_t^x$ given $S_t$ and $x_t$ |
| $\overline{V}_t^n(S_t^{x,n})$ | Approximation of the value of post-decision state $S_t^{x,n}$ |
| $\tilde{x}_t^n$ | Expected optimal decision within the current state |
| $\hat{v}_t^n$ | Approximation of the value of the current state given $\tilde{x}_t^n$ |
| $U^V(\overline{V}_{t-1}^{n-1}(S_{t-1}^{x,n}), S_{t-1}^{x,n}, \hat{v}_t^n)$ | Function to update the approximation $\overline{V}_{t-1}^n(S_{t-1}^{x,n})$ |
| $\alpha$ | Stepsize used in the updating function $U^V$ |
| $n = 1, \ldots, N$ | Iteration counter |
| $M$ | Number of iterations between "side simulations" |
| $O$ | Number of iterations of a "side simulation" |
| $K$ | Number of replications over all iterations $N$ |
| $a \in \mathcal{A}$ | Features |
| $\theta_a^n$ | Weight of feature $a$ |
| $\phi_a\left(S_t^{x,n}\right)$ | Basis function corresponding to feature $a$ |

Table 3.1: Frequently used notation throughout this chapter

## 3.2 The Nomadic Trucker Example

First, we briefly introduce the problem in Sect. 3.2.1 after which we present the MDP model (Sect. 3.2.2) and the ADP approach (Sect. 3.2.3).

### 3.2.1 Problem Introduction

The nomadic trucker problem is a stylized transportation problem in which a single trucker observes demands that arise randomly in different locations and he moves between these locations to accept those loads that maximize the long-term reward. This problem, which is similar to the well known taxicab problem, is described in [3, 4], and [10]. Here, we slightly modify the problem settings to allow repeatability of the experiments without having to provide extensive data sets.

Our trucker is characterized by its current location $l_t \in \mathcal{L}$ (set of 256 locations), the day of the week $d_t \in \{1,\ldots,7\}$ (Monday till Sunday), and its trailer type $k_t \in \{1,\ldots,3\}$ (small, medium, and large trailer). Every day, the driver observes loads to move from its current location to another location. The daily decision from a given location $i$ is which location $j$ to visit next, either through a loaded move (when a load from $i$ to $j$ is available) or an empty move, or to stay at the current location. After the decision, loads that are not selected are lost (assumed to be picked up by others). Further, it is assumed that all moves take less then a day, i.e., the next day a new decision has to be made.

The probability that, on a given day of the week $d$, a load from $i$ to $j$ will appear is given by $p_{ij}^d$ (see the Appendix). The trailer type attribute varies in a cyclic fashion, irrespective of the remaining attributes. For example, if at time period $t$ the trailer type attribute is large, then at time $t+1$ the trailer type will be small, and at time $t+2$ it will be medium. A larger trailer type results in higher rewards when traveling loaded or costs when traveling empty. We use the rewards/costs $c(k) = (1, 1.5, 2)$ per unit distance, for $k = 1,\ldots,3$. The rewards for loads leaving location $i$ are further multiplied by the origin probability $b_i$. The distance between $i$ and $j$ is given by $d(i, j)$.

We denote the described instance where there driver is characterized by a location, trailer type, and day of the week as the multi-attribute version. For the single-attribute version, we omit the trailer type and day of the week and use the settings for trailer type 1 and day of the week 1, i.e., $c(k) = 1$ and $p^d = 1$.

### 3.2.2 MDP Model

We subsequently present the following elements of the MDP model: the state (Sect. 3.2.2.1), the decision (Sect. 3.2.2.2), the costs (Sect. 3.2.2.3), the new information and transition function (Sect. 3.2.2.4), and the solution (Sect. 3.2.2.5).

#### 3.2.2.1 State

The state $S_t$ consists of resource and demand information: $S_t = \{R_t, D_t\}$. $R_t$ is a vector of attributes $(l_t, d_t, k_t)$ representing the current location of the trucker, the current day of the week, and the trailer type, respectively. $D_t$ is a vector indicating for each location $i \in \mathcal{L}$ whether there is a load available from $l_t$ to $i$ ($D_{t,i} = 1$) or not ($D_{t,i} = 0$). The state contains all the information necessary to make decisions; in this case the resource and demand information. The size of the state space is given by the number of possible settings of the resource vector $R_t$, which is 5376 ($256 \times 7 \times 3$), times the number of possible load realizations, which is $2^{256}$.

#### 3.2.2.2 Decision

The decision $x_t$ provides the location $j$ where we want to go to. Note that, given the used cost structure, if we decide to go from $l_t$ to $j$ and there is a load available from $l_t$ to $j$, it does not make sense to travel empty. In other words, from the demand vector $D_t$ we can infer whether the decision to go to location $j$ will imply an empty or a loaded move. Hence, it is sufficient to describe the decision $x_t$ with the location $j$, meaning the decision space $\mathcal{X}_t$ equals $\mathcal{L}$, with size 256.

#### 3.2.2.3 Costs

The costs of decision $x_t$ are given by

$$C(S_t, x_t) = \begin{cases} -c(k_t)d(l_t, x_t), & \text{if } D_{t,x_t} = 0. \\ c(k_t)d(l_t, x_t)b_i, & \text{if } D_{t,x_t} = 1. \end{cases} \tag{3.5}$$

#### 3.2.2.4 New Information and Transition Function

After making decision $x_t$ and before arrival in state $S_{t+1}$, new information $W_{t+1}$ arrives. Here, the new information gives the load availability at time $t+1$. The transition from $S_t$ to $S_{t+1}$ is given by

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}), \tag{3.6}$$

where $l_{t+1} = x_t$, $d_{t+1} = d_t \pmod 7 + 1$, $k_{t+1} = k_t \pmod 3 + 1$, and $D_{t+1} = W_{t+1}$. The size of the outcome space is given by all possible load realizations: $2^{256}$.

#### 3.2.2.5 Solution

The objective in this example is to maximize profit. Therefore, we need to replace the minimization objective in (3.3) by a maximization objective. However, to be consistent in our presentation, we use the minimization term throughout this chapter.

Even for this toy problem, the state space and the outcome space is already large. To ease the computation, we solve the problem for all possible "resource states", i.e., for each resource state $R_t$ at each stage $t$, we calculate its expected value considering all possible load availabilities with their probabilities. This can be seen as a "post-decision" state as we introduce in Sect. 3.2.3.1. Once the values for all "resource states" are determined, we can easily derive the optimal decision from each "full" state using (3.4) together with the transition function (3.6), where the transition only considers the change from the "full" state to the "resource state", i.e., the new information $W_{t+1}$ does not play a role in this transition.

For the exact computation of the values of the "resource states", it is not necessary to evaluate all possible permutations of the load combinations $W_{t+1}$. Within a given resource state $R_t$, we can rank on forehand the $2 \times |\mathcal{L}| = 512$ possible decisions (loaded and empty moves). We then start from the best possible decision and multiply its corresponding probability (if the decision involves a loaded move, we use the probability of having the corresponding load available, otherwise we use a probability of one) with its value; with one minus the before mentioned probability, we consider the second best possible decision and so on. We sum up all probabilities times the values to compute the expected value under the optimal policy.

We compute the optimal solution for three cases: (1) the infinite horizon single-attribute case, (2) the infinite horizon multi-attribute case, and (3) the finite horizon single-attribute case. For the finite horizon case, we can easily compute the value functions using backwards dynamic programming with (3.3). For the infinite horizon cases, we use value iteration to determine the optimal values. For the multi-attribute case, we use as initial state $S_0 = (1,1,1)$ and for the single-attribute cases we use $S_0 = (1)$. Further, for the finite horizon case, we use a discount $\gamma = 1$ and a horizon length $T = 20$, and for the infinite cases we use $\gamma = 0.9$. The optimal values are: (1) 8364.31 for the infinite horizon single-attribute case, (2) 11,448.48 for the infinite horizon multi-attribute case, and (3) 17,491.95 for the finite horizon single-attribute case.

## *3.2.3 Approximate Dynamic Programming*

Even though the introduced version of the nomadic trucker problem is a simplified problem that can easily be solved exactly, it allows us to introduce the basics of ADP. We introduce the concept of a post-decision state (Sect. 3.2.3.1), the forward dynamic programming approach (Sect. 3.2.3.2), and the use of value function approximations (Sect. 3.2.3.3). We give a typical outline of an ADP algorithm and present experimental results throughout Sects. 3.2.3.2 and 3.2.3.3.

### 3.2.3.1 Post-decision State

The post-decision state, represented by $S_t^x$, is the state immediately after action $x_t$, but before the arrival of new information $W_{t+1}$. The information embedded in the post-decision state allows us to estimate the downstream costs. We can assign the expected downstream costs $\mathbb{E}^{\omega}\{V_{t+1}(S_{t+1}|S_t^x,\omega)\}$ to every post-decision state $S_t^x$, thereby eliminating the need to evaluate all possible outcomes $\omega$ for every action. Consider the following optimality equations:

$$V_{t-1}^x(S_{t-1}^x) = \mathbb{E}^{\omega}\{V_t(S_t|S_{t-1}^x,\omega)\} \tag{3.7}$$

$$V_t(S_t) = \min_{x_t \in \mathcal{X}_t} \left(C_t(S_t,x_t) + \gamma V_t^x(S_t^x)\right) \tag{3.8}$$

$$V_t^x(S_t^x) = \mathbb{E}^{\omega}\{V_{t+1}(S_{t+1}|S_t^x,\omega)\} \tag{3.9}$$

If we substitute (3.9) into (3.8), we obtain the standard form of Bellman's equation (3.4). However, if we substitute (3.8) into (3.7), we obtain the optimality equations around the post-decision state variable

$$V_{t-1}^x\left(S_{t-1}^x\right) = \mathbb{E}^{\omega}\left\{\min_{x_t \in \mathcal{X}_t}\left(C_t\left(S_t,x_t\right) + \gamma V_t^x(S_t^x|S_{t-1}^x,\omega)\right)\right\}. \tag{3.10}$$

The basic idea now is (1) to use the deterministic optimization problem of (3.8) to make decisions and (2) to use the resulting observations to update an estimate $\overline{V}_{t-1}^{n-1}(S_{t-1}^x)$ of $V_{t-1}^x(S_{t-1}^x)$ thereby approximating the expectation in (3.10). We update the estimates $\overline{V}_{t-1}^{n-1}(S_{t-1}^x)$ iteratively over a number of iterations $n$, each consisting of a Monte Carlo simulation of the random information $\omega$, which will be further explained in Sect. 3.2.3.2.

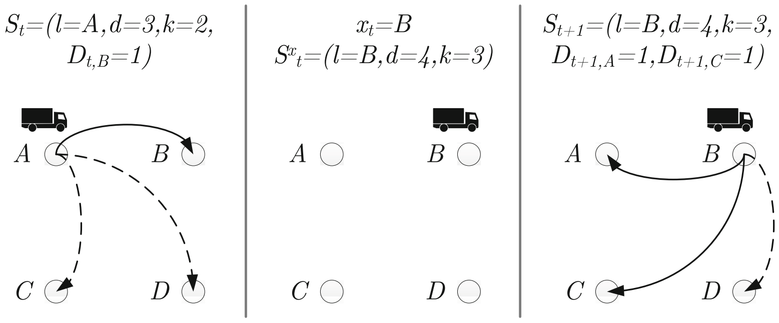
We express our transition from state $S_t$ with action $x_t$ to the post-decision state $S_t^x$ by

$$S_t^x = S^{M,x}(S_t,x_t). \tag{3.11}$$

For our example, the post-decision state $S_t^x$ is determined as if we had already arrived at the destination $x_t$ at time $t$. That is, we change the location, day of the week, and time components of the state to represent the day when we will be at the chosen location: $l_t^x = x_t$, $d_t^x = d_t \pmod 7 + 1$, and $k_t^x = k_t \pmod 3 + 1$. Note that, although the concept of a post-decision state is generally used in the context of ADP only, we already used it to calculate the exact solution of the MDP (see Sect. 3.2.2.5). An illustration of the transition of states can be found in Fig. 3.1.

### 3.2.3.2 Forward Dynamic Programming

In the forward dynamic programming algorithmic strategy, the Bellman's equations are solved only for one state at each stage, using estimates of the downstream values, and performing iterations $n$ to learn these downstream values. To make clear we are dealing with iterations, we add a superscript $n$ to the decisions and state variables.



$S_t$=(l=A,d=3,k=2, $D_{t,B}$=1)
$x_t$=B
$S^x{}_t$=(l=B,d=4,k=3)
$S_{t+1}$=(l=B,d=4,k=3, $D_{t+1,A}$=1,$D_{t+1,C}$=1)
A
B
C
D

Fig. 3.1: Transition of states in the nomadic trucker problem

We introduce the construct of approximated next-stage costs (estimated downstream values) $\overline{V}_t^n(S_t^{x,n})$, which replaces the standard expectation in Bellman's equations, see (3.9), with an approximation

$$\overline{V}_t^n(S_t^{x,n}) = \mathbb{E}^{\omega}\left\{V_{t+1}\left(S_{t+1}^n | S_t^{x,n}, \omega\right)\right\}. \tag{3.12}$$

Using the post-decision state and the approximated next-stage cost, the original Bellman's equations from (3.4) are converted to the ADP forward optimality equations, as seen in (3.13).

$$\hat{v}_t^n = \min_{x_t^n \in \mathcal{X}_t}\left(C(S_t^n, x_t^n) + \gamma \overline{V}_t^{n-1}\left(S^{M,x}(S_t^n, x_t^n)\right)\right). \tag{3.13}$$

The decision that minimizes (3.13) is given by

$$\tilde{x}_t^n = \underset{x_t^n \in \mathcal{X}_t}{\arg\min}\left(C(S_t^n, x_t^n) + \gamma \overline{V}_t^{n-1}\left(S^{M,x}(S_t^n, x_t^n)\right)\right). \tag{3.14}$$

Note that $\tilde{x}_t^n$ is a pure exploitation decision, i.e., the decision for which we currently expect it gives the best results. Given that the decision $\tilde{x}_t^n$ relies on the approximation $\overline{V}_t^{n-1}\left(S^{M,x}(S_t^n, x_t^n)\right)$, the decision might not be optimal with respect to the MDP solution. Further, as we will show later on, policies other than pure exploitation might be useful.

For each feasible decision $x_t^n$, there is an associated post-decision state $S_t^{x,n}$ obtained using (3.11). The ADP forward optimality equations are solved first at stage $t = 0$ for an initial state $S_0$, and then for subsequent stages and states until the end of the horizon for the finite horizon case, or a predetermined number of iterations for the infinite horizon case. In each iteration $n$, a sample path $\omega^n \in \Omega$ is drawn, with $\Omega$ being the set of all sample paths. We use $W_t(\omega^n)$ to denote the sample realization at time $t$ using the sample path $\omega^n$ in iteration $n$. To advance "forward" in time, from stage $t$ to $t+1$, the sample $W_{t+1}(\omega^n)$ is used. With this information, transition in the algorithm is done using the same transition as in the MDP model, see (3.6).

Immediately after the forward optimality equations are solved, the approximated next-stage cost $\overline{V}_t^{n-1}(S_t^{x,n})$ is updated retrospectively. The rationale behind this update is that, at stage $t$, the algorithm has seen new arrival information (via the simulation of $\omega^n$) and has taken a decision in the new state $S_t^n$ that incurs a cost. This means that the approximated next-stage cost that was calculated at the previous stage $t-1$, i.e., $\overline{V}_{t-1}^{n-1}(S_{t-1}^{x,n})$, has now been observed at stage $t$. To take advantage of this observation and improve the approximation, the algorithm updates this approximated next-stage cost of the previous post-decision state $S_{t-1}^{x,n}$ using the old approximation, i.e., $\overline{V}_{t-1}^{n-1}(S_{t-1}^{x,n})$ and the new approximation, i.e., the value $\hat{v}_t^n$ given by (3.13). We introduce $U^V$ to denote the process that takes all of the aforementioned parameters and "tunes" the approximating function as follows:

$$\overline{V}_{t-1}^{n}(S_{t-1}^{x,n}) \leftarrow U^V(\overline{V}_{t-1}^{n-1}(S_{t-1}^{x,n}), S_{t-1}^{x,n}, \hat{v}_t^n) \tag{3.15}$$

Using all ingredients mentioned in this section, the ADP algorithm consists of looping over iterations $n = 1, \ldots, N$, and within each iteration, sequentially solving a subproblem for each $t \in \mathcal{T}$, using sample realizations of $W_t$, and updating our approximation of 'future' costs with (3.15). Consecutively, the subproblems are solved using the updated value function approximations in the next iteration. The output of the algorithm is the approximation $\overline{V}_t^N(S_t^x)$, for all $t \in \mathcal{T}$, which can be used to find the best decision for each time period and each state.

A typical outline of an ADP algorithm is shown in Algorithm 1. The infinite horizon version of this algorithm is basically the same, except (1) all subscripts $t$ are removed, (2) $\omega^n$ represents a single sample instead of a sample path, (3) the loop over $t$ is removed, (4) the condition "if $t > 0$" is removed, and (5) the previous post-decision state is the one from the previous iteration, $S^{x,n-1}$.

**Algorithm 1** *Approximate Dynamic Programming algorithm*

```
Step 0. Initialization
    Step 0a. Choose an initial approximation $\overline{V}_t^0 \forall t \in \mathcal{T}$.
    Step 0b. Set the iteration counter $n = 1$, and set the maximum number of iterations $N$.
    Step 0c. Set the initial state to $S_0^1$.
Step 1. Choose a sample path $\omega^n$.
Step 2. Do for $t = 0, \ldots, T$:
    Step 2a. Solve (3.13) to get $\hat{v}_t^n$ and (3.14) to get $\tilde{x}_t^n$.
    Step 2b. If $t > 0$, then update the approximation $\overline{V}_{t-1}^n\left(S_{t-1}^{x,n}\right)$ for the previous post-
             decision $S_{t-1}^{x,n}$ state using (3.15).
    Step 2c. Find the post-decision state $S_t^{x,n}$ with (3.11) and the new pre-decision state $S_{t+1}^n$
             with (3.6).
Step 3. Increment $n$. If $n \leq N$ go to Step 1.
Step 4. Return the value functions $\overline{V}_t^N\left(S_t^{x,n}\right) \forall t \in \mathcal{T}, S_t \in \mathcal{S}$.
```

Algorithm 1 relies on classical approximate value iteration with a pure forward pass. This means that at each step forward in time in the algorithm, the value function approximations are updated. As the algorithm steps forward in time, it may take many iterations before the costs incurred in later time periods are correctly transferred to the earlier time periods. To overcome this, the ADP algorithm can also be used with a double pass approach [10], consisting of a forward pass and a backward pass. In the forward pass, we simulate decisions moving forward in time, remembering the trajectory of states, decisions, and outcomes. Then, in a backward pass, we update the value functions moving backwards in time using the trajectory information. For the double pass algorithm, we remove the computation of $\hat{v}_t^n$ from Step 2a, delete Step 2b, and add an extra step just before Step 3 (renaming original Step 3 and Step 4 by Step 4 and Step 5 respectively) given in Algorithm 2:

**Algorithm 2** *Backward pass to be used in the ADP algorithm*

```
Step 3. Do for $t = T, T-1, \ldots, 1$:
    Step 3a. Compute $\hat{v}_t^n$ using the decision $\tilde{x}_t^n$ from the forward pass:
        $\hat{v}_t^n = C_t(S_t^n, \tilde{x}_t^n) + \gamma \hat{v}_{t+1}^n$, with $\hat{v}_{T+1}^n = 0$.
    Step 3b. Update the approximation $\overline{V}_{t-1}^n\left(S_{t-1}^{x,n}\right)$ for the previous post-decision state
        $S_{t-1}^{x,n}$ using (3.15).
```

## Computational Results

We now illustrate the working of this basic algorithm using the three variants of the nomadic trucker problem: infinite horizon single-attribute, infinite horizon multi-attribute, and finite horizon single-attribute. For the updating function we use a fixed stepsize $\alpha = 0.05$ given by $U^V(\overline{V}_{t-1}^{n-1}(S_{t-1}^{x,n}), S_{t-1}^{x,n}, \hat{v}_t^n) = (1-\alpha)\overline{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \alpha \hat{v}_t^n$.

We show two performance measures of the ADP algorithm. First, we show the estimate $\overline{V}_0^n\left(S_0^{x,n}\right)$ of the initial state $S_0^{x,n}$ for different number of iterations $n$. Next, we show the discounted rewards of using the estimates for different number of iterations $n$. More precisely, for a given number of iterations $n$, we perform a simulation on the side. Each of these simulations uses $O$ iterations, fixing the value function estimates and following the policy that uses these values (basically following Algorithm 1 with the initial approximation $\overline{V}_0^n$ resulting from the past $n$ iterations and skipping Step 2b). We perform the simulation every $M$th iteration, i.e., for $n = M, 2M, \ldots, N$. Finally, to provide representative results, we repeat the whole procedure $K$ times and report the averages. The used settings for $N$, $M$, $O$, and $K$ are shown in the figure captions.

The results of the basic ADP algorithm is shown in Fig. 3.2 under the policy "Expl-F", since we follow the pure exploitation policy (3.13) with a fixed stepsize (F). In addition, we show the results using two other stepsizes. First, the harmonic stepsize (H) given by $\alpha^n = \max\left\{\frac{\lambda}{\lambda+n-1}, \alpha^0\right\}$, with $\lambda = 25$ and $\alpha^0 = 0.05$. Second, the BAKF stepsize (B), the bias adjusted Kalman Filter also known as OSA (Optimal Stepsize Algorithm). For a review on stepsizes we refer to [3] and [10, Chap. 11]. The policy "OPT" refers to the optimal value.

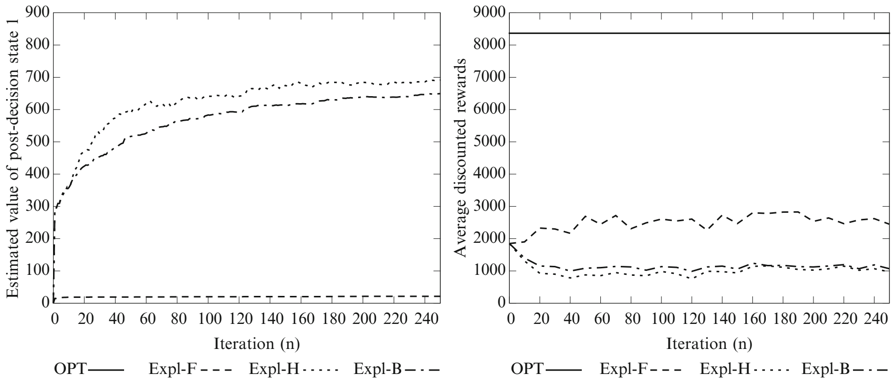


Fig. 3.2: Infinite horizon single-attribute case: resulting estimate $\overline{V}_0^n\left(S_0^{x,n}\right)$ (*left*) and realized rewards (*right*), using $N = 250$, $M = 10$, $O = 1000$, and $K = 100$

Clearly, the value function estimate for the initial state is still far off: more then 90% away from the optimal value. Especially, the fixed stepsize gives terrible results. The explanation is that because we initialized all values at zero, the first observation only increases the estimate with 5% of the observed value. The other two stepsizes start with $\alpha^1 = 1$, resulting in a faster increase in estimated value. However, when we look at the performance, the "better" value function estimates result in worse performance. The explanation is that after a couple of iterations, some states have been visited more often than other states. As a result, for some states we might still use the initialized approximations (in our case values of zero) whereas for other states we have a reasonable estimate. When making the pure exploitation decision, we now tend to visit the states we have visited before, even if this would result in relatively high direct costs. In this case, we perform worse compared to a myopic policy. The results for following the myopic policy can be found at $n = 0$ in the right figure, since we then perform a simulation with the initial approximation of having zero downstream costs. Although the results are caused by some simplifications within the used ADP algorithm, the resulting phenomenon might also be present in more advanced ADP implementations, since the decision what state to measure next might still be biased by the number of measurements of the different states.

In general, using the ADP algorithm as shown before would not work. Most of the time, we need to make some modifications. First, within our ADP algorithm we need to make a tradeoff between exploitation, i.e., making the best decision based on the current value function estimates using (3.13), and exploration to learn the value of states frequented less often. Second, we need a value function approximation that is able to generalize across states, i.e., an observation not only results in an update for the value of the corresponding state, but also of other states. In the remainder of this section, we briefly touch upon the exploration issue. The issue of having a better value function approximation is discussed in Sect. 3.2.3.3.

To overcome the problem of limited exploration, we might enforce exploration by stating that a certain fraction of the time, say $\varepsilon$, the policy should perform exploration using a random decision instead of exploitation using the decision $\tilde{x}_t^n$ from (3.14). This decision policy is known as $\varepsilon$-greedy. When making an exploration decision, it is likely that $x_t^n \neq \tilde{x}_t^n$. We now have a choice whether we use $\hat{v}_t^n$ (corresponding to decision $\tilde{x}_t^n$) to update $\overline{V}_{t-1}^n\left(S_{t-1}^{x,n}\right)$ or to use the estimated costs corresponding with the actual decision $x_t^n$, i.e., $C\left(S_t^n, x_t^n\right) + \gamma \overline{V}_t^{n-1}\left(S^{M,x}\left(S_t^n, x_t^n\right)\right)$. The policy to determine the decisions on what state to visit next is often referred to as the behavior or sampling policy. The policy that determines the decision that seems to be the best, i.e., using (3.14) is denoted by learning policy. When the sampling policy and the learning policy are the same, this is called on-policy learning, otherwise it is called off-policy learning. In most cases, off-policy learning results in faster convergence; but there are cases where on-policy learning is preferred, see [10, Chap. 9] for more information. In the remainder, unless stated differently, we use off-policy learning.

The results of the $\varepsilon$-greedy policies are shown in Fig. 3.3 under the policy "Eps$\varepsilon$-S", where $\varepsilon = 0.25$ or $\varepsilon = 1$, and $S$ denotes the stepsize (H for harmonic and B for BAKF). The policies Heps$\varepsilon$ will be introduced later on.

From Fig. 3.3, we see that the $\varepsilon$-greedy policies improve the performance, both with respect to convergence of value functions (estimated values of the initial post-decision state) and the average discounted rewards, when compared to the exploitation policies. However, we also see that policies with faster convergence in value function not necessarily yield better performance. Again, this phenomenon will also be present at more advanced ADP implementations. First, having a good approximation for one state does not necessarily mean we have a good approximation for other states. Particularly when using value function approximations that are able to generalize across states, we might have states that we consistently underestimate and others that we consistently overestimate. Second, the (relative) ordering of states might already result in good performance of the policy (i.e., decision function) itself. In this example, the absolute values of the downstream costs might be less important when choosing between decisions with similar direct costs.

Still, we observe that our estimate is far off from the optimal value and we achieve only about 68% of the rewards under the optimal policy. To further improve the learning rate, i.e., increase the performance using the same number of iterations,

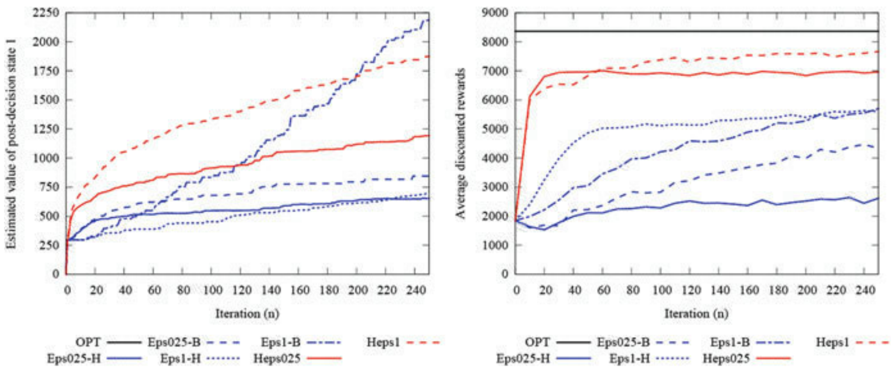


Fig. 3.3: Infinite horizon single-attribute case: resulting estimate $\overline{V}_0^n\left(S_0^{x,n}\right)$ (*left*) and realized rewards (*right*), using $N = 250$, $M = 10$, $O = 1000$, and $K = 100$

we are going to use a Value Function Approximation (VFA) that is able to generalize across states. There are many options for this, in this chapter we present two specific forms: hierarchical state aggregation and a basis function approach. For the nomadic trucker problem, we illustrate the state aggregation approach.

### 3.2.3.3 Value Function Approximation

Although adopting the concept of the post-decision state greatly reduced the computational burden, (3.10) still requires a post-decision state to be visited sufficiently often in order to learn about its associated downstream costs, which would not be possible for realistic sized problem instances. The reason for this is that the value function approximation used in the previous section is updated for one state per stage per iteration. This approach is known as the lookup-table approach. A good value function approximation is able to generalize across states, such that an observation for one state results in an update of the value of many states.

For this problem, we use an hierarchical aggregation approach as presented in [4]. A standard strategy in ADP is to aggregate the state space. Each state belongs to an aggregated state, and instead of estimating the value of states, we estimate the value of aggregate states consisting of multiple states. However, aggregation requires resolving the classic tradeoff between aggregation error and sampling error. In the hierarchical aggregation approach, we use multiple aggregation levels and each observation is used to update aggregate states at different aggregation level. The value of a state is estimated using a weighted combination of the value of the aggregated states this state belongs to.

For this example, we use the following hierarchical aggregation structure. With respect to the state variables trailer type $k$ and day of the week $d$, we either include them or leave them out. With respect to the location, we use a structure with on the lowest level the $16 \times 16$ locations, one level up we group sets of 4 locations, resulting in $8 \times 8$ aggregated locations, and so on. We perform aggregation on one state variable at a time, achieving an almost exponential decline in the size of the state space. An overview of the aggregation structure is given in Table 3.2, where a '*' corresponds to a state variable included in the aggregation level and a '–' indicates that it is aggregated out.

Table 3.2: Hierarchical aggregation structure

| Level | Location | Trailer type | Day of the week | Size of the state space |
|---|---|---|---|---|
| 0 | $(16 \times 16)$ | * | * | $256 \times 3 \times 7 = 5376$ |
| 1 | $(8 \times 8)$ | * | * | $64 \times 3 \times 7 = 1344$ |
| 2 | $(4 \times 4)$ | * | * | $16 \times 3 \times 7 = 336$ |
| 3 | $(4 \times 4)$ | – | * | $16 \times 1 \times 7 = 112$ |
| 4 | $(2 \times 2)$ | – | * | $4 \times 1 \times 7 = 28$ |
| 5 | – | – | * | $1 \times 1 \times 7 = 7$ |
| 6 | – | – | – | $1 \times 1 \times 1 = 1$ |

## Computational Results

The results of using the hierarchical aggregation approach are shown in Fig. 3.3, policy $Heps\varepsilon$, with $\varepsilon = 0.25$ and $\varepsilon = 1$. This policy does not require a stepsize, since this is included in the updating equations of the hierarchical aggregation approach. Clearly, the $Heps\varepsilon$ policies converge faster to the optimal values. The policy Heps1 results in only 8% lower profits compare to the optimal policy.

Finally, to gain insight into the long term performance, we show the results using 25,000 iterations, but with fewer replications ($K = 10$), in Fig. 3.4. After 25,000 measurements, the policies Eps1-B and Heps1-B are both within 1% of the optimal performance.

Next, we show the results for the finite horizon case in Fig. 3.5. Here, we both consider the single pass (SP) and the double pass (DP) version of the ADP algorithm. Here, the pure exploitation policy does not benefit from a double pass, simply because with the double pass, the decisions will be even more biased towards states visited before. The same also holds for the $\varepsilon$-greedy policies, since they explore only 5% of the time. However, the hierarchical $\varepsilon$-greedy policies do benefit from the double pass. In addition, the hierarchical $\varepsilon$-greedy policies also benefit from exploration (Heps005). With increasing number of measurements, the hierarchical $\varepsilon$-greedy policies are eventually outperformed by the single pass $\varepsilon$-greedy policies (Heps005-Double 2.56% away from optimum and Eps005-Single 1.56% away from optimum). However, using 25,000 measurements is not representative for a problem

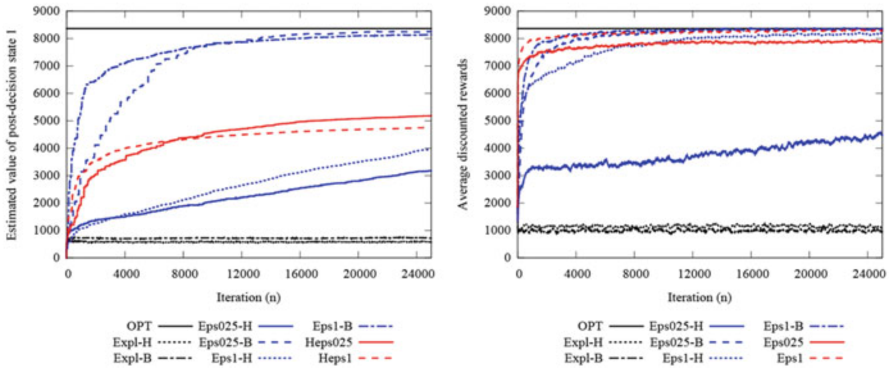


Fig. 3.4: Infinite horizon single-attribute case: resulting estimate $\overline{V}_0^n\left(S_0^{x,n}\right)$ (*left*) and realized rewards (*right*), using $N = 25{,}000$, $M = 10$, $O = 1000$ and $K = 10$. For the rewards resulting from the simulations, the 2500 observations are smoothed using a window of 10

having a state space with size 256. In most ADP applications, the number of iterations would be only a fraction of the size of the state space, say a couple of hundred in this example. Clearly, in these cases the hierarchical aggregation approach performs best. The results after 200 measurements, for various policies, including on-policy and off-policy variations, can be found in Fig. 3.6.

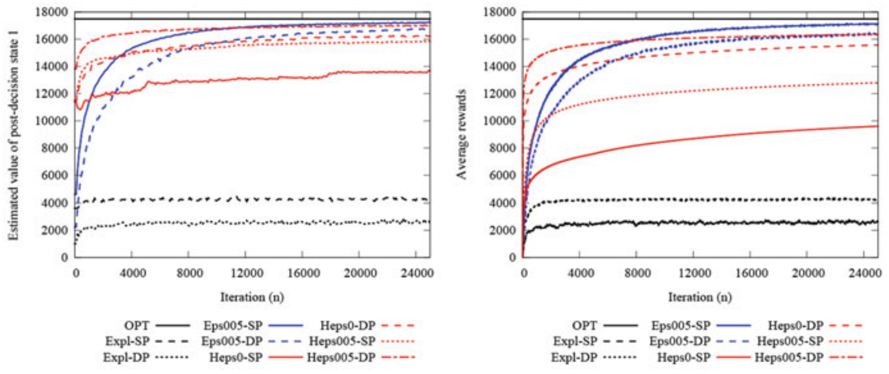


Fig. 3.5: Finite horizon single-attribute case: resulting estimate $\overline{V}_0^n\left(S_0^{x,n}\right)$ (*left*) and realized rewards (*right*), using $N = 25{,}000$, $M = 100$, $O = 100$ and $K = 10$. For the exploitation and $\varepsilon$-greedy policies, the BAKF stepsize is used

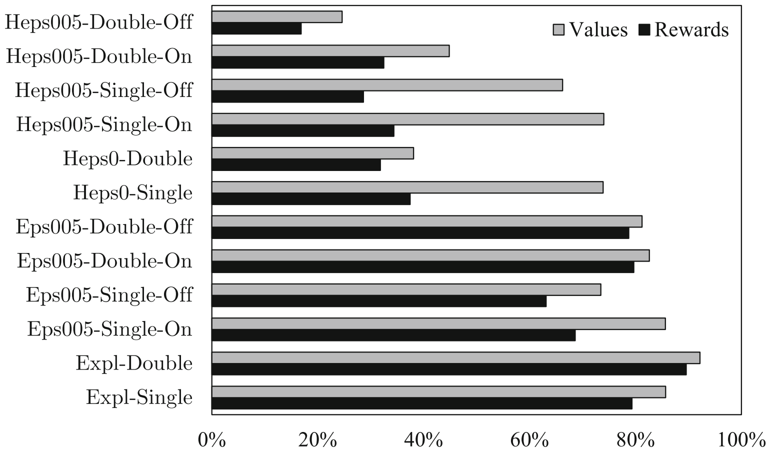


Fig. 3.6: Finite horizon single-attribute case: deviation of the estimate $\overline{V}_0^n\left(S_0^{x,n}\right)$ and realized rewards from the optimal values, using $N = 200$, $O = 100$ and $K = 10$

The results for the multi-attribute case can be found in the Appendix. The results are similar to the those observed for the single-attribute case. One remarkable behavior is that the $\varepsilon$-greedy policy shows an initial decline in performance after which it improves. Again, this is caused by the fact that the decisions are biased towards visiting states that have been measured before, resulting in relatively high direct costs. Once the value of enough states are known, the performance improves. Obviously, this behavior is not visible for the Hierarchical $\varepsilon$-greedy policy since it is able to generalize across states.

## 3.3 A Freight Consolidation Example

First, we briefly introduce the problem in Sect. 3.3.1 after which we present the MDP model (Sect. 3.3.2) and the ADP approach (Sect. 3.3.3).

### *3.3.1 Problem Introduction*

We now consider another transportation example, with completely different characteristics. We use this example to (1) illustrate the typical use of ADP for resource allocation problems and (2) to illustrate a value function approximation relying on the basis function approach.

We consider the planning problem that arises when a company transports freights from a single origin to different destinations, periodically, using a high capacity mode. The destinations of these freights are far away and closer among themselves than to the origin of transportation. For this reason, the long-haul is the same in every trip, independent of which freights were consolidated at the origin. However, the last-mile route varies according to the destinations of the freights that were consolidated at the beginning of the long-haul. In addition, there is an alternative, low capacity mode that can be used to transport freights directly from the origin to their destination. Each period, the choice is which freights to allocate in the current period to the high capacity mode, which ones to transport with the low capacity mode, and which ones to postpone to a later period. The costs of the long-haul are fixed, but the last-mile costs depend on the combination of destinations visited. The costs per freight for the low capacity mode are considerably higher than the high capacity mode. The objective of the company is to reduce its total costs over time and to use the long-haul, high capacity mode capacity efficiently. Properly balancing the consolidation and postponement of freights, such that only a few close-by destinations are visited each day, is therefore a challenge for the company but also a necessity for its efficient operation.

We consider a dynamic multi-period long-haul freight consolidation problem where decisions are made on consecutive periods $t$ over a finite horizon $\mathcal{T} = \{0,1,2,\ldots,T^{max}-1\}$. For simplicity, we refer to a period as a day in the remainder of this example. Each freight must be delivered to a given destination $d$ from a group of destinations $\mathcal{D}$ within a given time-window. The time-window of a freight begins at a release-day $r \in \mathcal{R} = \{0,1,2,\ldots,R^{max}\}$ and ends at a due-day $r+k$, where $k \in \mathcal{K} = \{0,1,2,\ldots,K^{max}\}$ defines the length of the time-window. The arrival-day $t$ of a freight is the moment when all its information is known to the planner. Note that $r$ influences how long the freights are known before they can be transported, and thus influences the degree of uncertainty in the decisions.

New freights become available as time progresses. These freights and their characteristics follow a stochastic arrival process. Between two consecutive days, a number of freights $f$ arrive with probability $p^F_f$, independent of the arrival day. Each freight has destination $d$ with probability $p^D_d$, release-day $r$ with probability $p^R_r$, and time-window length $k$ with probability $p^K_k$, independent of the day and of other freights.

Each day, there is only one long-haul vehicle which transports at most $Q$ freights. Its cost is $C_{\mathcal{D}'}$, where $\mathcal{D}' \subseteq \mathcal{D}$ denotes the subset of destinations visited. There is also an alternative transport mode for each destination $d$, which can only be used for freights whose due-day is immediate (i.e., $r = k = 0$). The cost of the alternative transport mode is $B_d$ per freight to destination $d$, and there is no limit on the number of freights that can be transported using this mode.

## 3.3.2 MDP Model

We subsequently present the following elements of the MDP model: the state (Sect. 3.3.2.1), the decision (Sect. 3.3.2.2), the costs (Sect. 3.3.2.3), the new information and transition function (Sect. 3.3.2.4), and the solution (Sect. 3.3.2.5).

#### 3.3.2.1 State

At each time period $t$, there are known freights with different characteristics. We define $F_{t,d,r,k}$ as the number of known freights at stage $t$, whose destination is $d$, whose release-day is $r$ stages after $t$, and whose time-window length is $k$ (i.e., its due-day is $r+k$ stages after $t$). The state of the system at stage $t$ is denoted by $S_t$ and is defined as the vector of all freight variables $F_{t,d,r,k}$, as seen in (3.16).

$$S_t = \left[F_{t,d,r,k}\right]_{\forall d \in \mathcal{D}, r \in \mathcal{R}, k \in \mathcal{K}} \tag{3.16}$$

#### 3.3.2.2 Decision

The decision at each stage is which released freights (i.e., freights with $r = 0$) to consolidate in the long-haul vehicle. We use the integer variable $x_{t,d,k}$ as the number of freights that are consolidated in the long-haul vehicle at stage $t$, which have destination $d$ and are due $k$ stages after $t$. We denote the vector of decision variables at stage $t$ as $x_t$. Due to the time-window of freights, the possible values of these decision variables are state dependent. Thus, the feasible space of decision vector $x_t$, given a state $S_t$, is as follows:

$$x_t = \left[x_{t,d,k}\right]_{\forall d \in \mathcal{D}, k \in \mathcal{K}} \tag{3.17a}$$

$$\text{s.t.}$$

$$\sum_{d \in \mathcal{D}} \sum_{k \in \mathcal{K}} x_{t,d,k} \leq Q, \tag{3.17b}$$

$$0 \leq x_{t,d,k} \leq F_{t,d,0,k} \tag{3.17c}$$

#### 3.3.2.3 Costs

The cost of a decision $x_t$ at a state $S_t$ depends on the destinations visited by the long-haul vehicle and the use of the alternative mode (i.e., $C_{\mathcal{D}'}$ and $B_d$, respectively). The costs at stage $t$ are given by $C(S_t, x_t)$. From the decision $x_t$, we derive the combination of terminals that will be visited by the high capacity mode (which determines the high capacity vehicle costs) as well as the number of urgent freights that are not scheduled to be delivered by the high capacity mode (which determines the low capacity vehicle costs).

#### 3.3.2.4 New Information and Transition Function

We introduce a single arrival information variable $\widetilde{F}_{t,d,r,k}$, which represents the freights that arrived from outside the system between stages $t-1$ and $t$, with destination $d$, release-day $r$, and time-window length $k$. We denote the vector of all arrival information variables at stage $t$ as $W_t$, as seen in (3.18).

$$W_t = \left[\widetilde{F}_{t,d,r,k}\right]_{\forall d \in \mathcal{D}, r \in \mathcal{R}, k \in \mathcal{K}} \tag{3.18}$$

The consolidation decision $x_t$ and arrival information $W_t$ have an influence on the transition of the system between stages $t-1$ and $t$. In addition, the relative time-windows have an influence on the transition between related freight variables. To represent all of these relations, we use the following transition function:

$$S_t = S^M\left(S_{t-1}, x_{t-1}, W_t\right) | t > 0 \tag{3.19a}$$

s.t.

$$F_{t,d,0,k} = F_{t-1,d,0,k+1} - x_{t-1,d,k+1} + F_{t-1,d,1,k} + \widetilde{F}_{t,d,0,k}\ ,\ k < K^{max} \tag{3.19b}$$

$$F_{t,d,0,K^{max}} = F_{t-1,d,1,K^{max}} + \widetilde{F}_{t,d,0,K^{max}} \tag{3.19c}$$

$$F_{t,d,r,k} = F_{t-1,d,r+1,k} + \widetilde{F}_{t,d,r,k}\ ,\ 1 \leq r < R^{max} \tag{3.19d}$$

$$F_{t,d,R^{max},k} = \widetilde{F}_{t,d,R^{max},k} \tag{3.19e}$$

For the transition of the freight variables $F_{t,d,r,k}$ in (3.19a), we distinguish between four cases. First, freights which are already released at stage $t$ (i.e., $r = 0$) and have a time-window length of $k < K^{max}$ are the result of: (1) freights from the previous stage $t-1$ which were already released, had time-window length $k+1$, and were not transported (i.e., $F_{t-1,d,0,k+1} - x_{t-1,d,k+1}$), (2) freights from the previous stage $t-1$ with next-stage release-day (i.e., $r = 1$) and time-window length $k$ (i.e., $F_{t-1,d,1,k}$), and (3) the new (random) arriving freights with the same characteristics (i.e., $\widetilde{F}_{t,d,0,k}$) as seen in (3.19b). Second, freights that are already released at day $t$ and have a time-window length $k = K^{max}$ are the result of freights from the previous stage $t-1$ that had a next day release and the same time-window length (i.e., $F_{t-1,d,1,K^{max}}$), in addition to the freights that arrived between the previous and the current day with the same characteristics (i.e., $\widetilde{F}_{t,d,0,K^{max}}$), as seen in (3.19c). Third, freights which are released at stage $t$ (i.e., $r \geq 1$) are the result of: (1) freights from the previous stage $t-1$ with a release-day $r+1$ and that have the same time-window length $k$, and (2) the new freights with the same characteristics (i.e., $\widetilde{F}_{t,d,r,k}$), as seen in (3.19d). Fourth, freights which have the maximum release-day (i.e., $r = R^{max}$) are the result only of the new freights with the same characteristics (i.e., $\widetilde{F}_{t,d,R^{max},k}$), as seen in (3.19e).

#### 3.3.2.5 Solution

Again, the formal objective of the model is to find the policy $\pi \in \Pi$ that minimizes the expected costs over the planning horizon, given an initial state $S_0$, as seen in (3.1). Following Bellman's principle of optimality, the best policy $\pi$ for the planning horizon can be found solving a set of stochastic recursive equations that consider the current-stage and expected next-stage costs, as seen in (3.3). We can solve (3.3) plugging in the transition function (3.19a) and specifying the probability $\mathbb{P}(W_{t+1} = \omega)$, which can be found in [9].

Naturally, only very small problem instances can be solved to optimality. The instance we use in this example has the following characteristics. We consider a planning horizon of a working week ($T^{max} = 5$), three destinations, ($|\mathcal{D}| = 3$), one release-day ($R^{max} = 0$), three time-window lengths ($K^{max} = 2$), and at most two freights per day ($|\mathcal{F}| = 2$). The capacity of the long-haul, high capacity vehicle is $Q = 2$. All probabilities and costs are given in the Appendix.

The given problem settings result in an MDP model with 2884 states. For simplicity, we choose to explain more into detail two of these states. The first state, refereed to as "State 1" has only one freight for destination 2 with a time-window length of 2 (i.e. $F_{0,2,0,2} = 1$). The second state, referred to as "State 2", has a total of six freights: one urgent freight for destination 2 and 3, three freights for destination 2 with time-window length 1, and one freight for destination 2 with time-window length 2 (i.e., $F_{0,2,0,0} = F_{0,3,0,0} = 1$, $F_{0,2,0,1} = 3$, $F_{0,2,0,2} = 1$). The optimal costs for State 1 and State 2 are 968.15 and 2619.54, respectively. We choose these two states to show, in the following, the different design challenges arising when applying the ADP algorithm with basis functions to different initial states.

## *3.3.3 Approximate Dynamic Programming*

The ADP approach is presented using a similar setup as with the first example. We subsequently present the post-decision state (Sect. 3.3.3.1), the forward dynamic programming approach (Sect. 3.3.3.2), and the use of value function approximations (Sect. 3.3.3.3).

### 3.3.3.1 Post-decision State

The post-decision state contains all post-decision freight variables $F^{x,n}_{t,d,r,k}$:

$$S^{x,n}_t = \left[F^{x,n}_{t,d,r,k}\right]_{\forall d \in \mathcal{D}, r \in \mathcal{R}, k \in \mathcal{K}} \tag{3.20}$$

We use the following function $S^{M,x}$ for the transition from the state $S^n_t$ to the post-decision state $S^{x,n}_t$:

$$S^{x,n}_t = S^{M,x}\left(S^n_t, x^n_t\right) \tag{3.21a}$$

$$\text{s.t.}$$

$$F^{x,n}_{t,d,0,k} = F^n_{t,d,0,k+1} - x^n_{t,d,k+1} + F^n_{t,d,1,k}\ ,\ k < K^{max} \tag{3.21b}$$

$$F^{x,n}_{t,d,0,K^{max}} = F_{t-1,d,1,K^{max}} \tag{3.21c}$$

$$F^{x,n}_{t,d,r,k} = F^n_{t,d,r+1,k}\ ,\ 1 \leq r < R^{max} \tag{3.21d}$$

This function works in the same way as the MDP transition function defined in (3.19a), with the difference that the new arrival information $W_t$ is not included. An illustration of the transition of states can be found in Fig. 3.7.

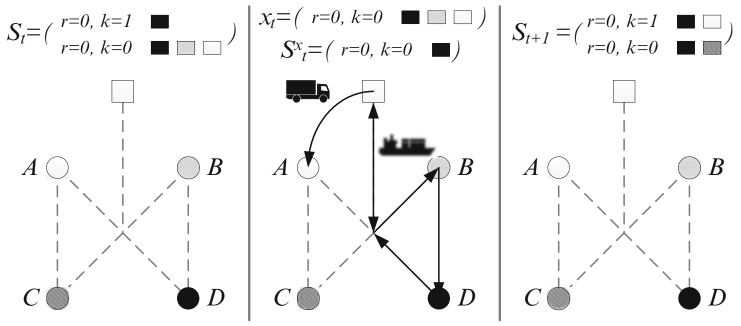


Fig. 3.7: Transition of states in the freight consolidation problem

### 3.3.3.2 Forward Dynamic Programming

We can directly apply Algorithm 1 using the lookup-table approach. In a minimization problem, initializing the values of the lookup-table with zero will automatically result in an "exploration policy". In such an initialization, a post-decision state that has not been visited before is more attractive (zero downstream costs) than one that has been visited before and has resulted in some costs. In our example, we choose to initialize the values to zero to take advantage of the exploration behavior. Furthermore, we use the harmonic stepsize (see Sect. 3.2.3.2) with $\lambda = 25$ and $\alpha^0 = 0.05$. The ADP runs for a total of 250 iterations, using the double pass approach. The estimated values (i.e., learned costs) for State 1 and State 2 are 1153.85 and 2814.74, respectively. These values are 19% and 7% higher then the optimal MDP costs, respectively. The average costs for State 1 and State 2 (obtained through a simulation of the policy resulting from the learned values) are 1550.65 and 2852.75, respectively. These average costs are 19% and 9% higher then the optimal MDP costs, respectively. In the following, we elaborate on how the performance of the value function approximation can be improved through the use of basis functions.

### 3.3.3.3 Value Function Approximation

For this example, we introduce a frequently used approximation strategy using basis functions. An underlying assumption in using basis functions is that particular features, or quantitative characteristics, of a (post-decision) state can be identified, that explain, to some extent, what the value of that post-decision state is. In our problem, features such as the number of urgent freights, the number of released freights that are not urgent, and the number of freights that have not been released for transport, can explain part of the value of a post-decision state. Basis functions are then created for each individual feature to quantify the impact of the feature on the value function.

We define a set of features $\mathcal{A}$ for which the value of each feature $a \in \mathcal{A}$ of a post-decision state $S_t^{x,n}$ is obtained using a basis function $\phi_a(S_t^{x,n})$. We assume the approximated next-stage value of a post-decision state can be expressed by a weighted linear combination of the features, using the weights $\theta_a^n$ for each feature $a \in \mathcal{A}$, as follows:

$$\overline{V}_t^{x,n}\left(S_t^{x,n}\right) = \sum_{a \in \mathcal{A}} \theta_a^n \phi_a\left(S_t^{x,n}\right) \tag{3.22}$$

The weight $\theta_a^n$ is updated recursively in each iteration $n$. Note that (3.22) is a linear approximation, as it is linear in its parameters. The basis functions themselves can be nonlinear [10].

The use of features and weights for the approximating the value function $\overline{V}_t^n(S_t^{x,n})$ is comparable to the use of regression models for fitting data to a (linear) function. In that sense, the independent variables of the regression model would be the features of the post-decision state and the dependent variable would be the value of the post-decision state. However, in contrast to regression models, the data in our ADP is generated iteratively inside an algorithm and not all at once. Therefore, the updating process $U^V$ for the approximating function in (3.22) cannot be based only on solving systems of equations as in traditional regression models.

Several methods are available to “fine-tune” the weights $\theta_a^n$ for each feature $a \in \mathcal{A}$ after each iteration. An effective approach is the recursive least squares method, which is a technique to compute the solution to a linear least squares problem [10]. Two types of recursive least squares methods are available. The least squares method for *nonstationary* data provides the opportunity to put increased weight on more recent observations, whereas the least squares method for *stationary* data puts equal weight on each observation. For the purpose of learning the weights within an ADP algorithm, the recursive least squares method for nonstationary data is more appropriate. The method for updating the value function approximations with the recursive least squares method for nonstationary data is explained in detail in [10]. Nevertheless, the equations used in this method are given below.

The weights $\theta_a^n$, for all $a \in \mathcal{A}$, are updated each iteration ($n$ is the iteration counter) by

$$\theta_a^n = \theta_a^{n-1} - H_n \phi_a\left(S_t^{x,n}\right)\left(\overline{V}_{t-1}^{n-1}\left(S_{t-1}^{x,n}\right) - \widehat{v}_t^n\right),$$

where $H^n$ is a matrix computed using

$$H^n = \frac{1}{\gamma^n} B^{n-1},$$

and where $B^{n-1}$ is an $|\mathcal{A}|$ by $|\mathcal{A}|$ matrix that is updated recursively using

$$B^n = \frac{1}{\alpha^n}\left(B^{n-1} - \frac{1}{\gamma^n}\left(B^{n-1}\phi\left(S_t^{x,n}\right)\left(\phi\left(S_t^{x,n}\right)\right)^T B^{n-1}\right)\right).$$

The expression for $\gamma^n$ is given by

$$\gamma^n = \alpha^n + \phi\left(S_t^{x,n}\right)^T B^{n-1} \phi\left(S_t^{x,n}\right).$$

$B^n$ is initialized by using $B^0 = \varepsilon I$, where $I$ is the identity matrix and $\varepsilon$ is a small constant. This initialization works well when the number of observations is large [10]. The parameter $\alpha^n$ determines the weight on prior observations of the value. Setting $\alpha^n$ equal to 1 for each $n$ would set equal weight on each observation, and implies that the least squares method for stationary data is being used. Setting $\alpha^n$ to values between 0 and 1 decreases the weight on prior observations (lower $\alpha^n$ means lower weight). We define the parameter $\alpha^n$ by

$$\alpha^n = \begin{cases} 1 & \text{, stationary} \\ 1 - \frac{\delta}{n} & \text{, nonstationary} \end{cases} \tag{3.23}$$

where $1 - \frac{\delta}{n}$, with $\delta = 0.5$, is a function to determine $\alpha_n$ that works well in our experiments.

For this example, there are a number of possible features, such as:

1. Each state variable: number of freights with specific attributes.
2. Per destination, the number of freights that are not yet released for transport (i.e., future freights).
3. Per destination, the number of freights that are released for transport and whose due-day is not immediate (i.e., may-go freights).
4. Per destination, a binary indicator to denote the presence of urgent freights (i.e., must-visit destination).
5. For each state variable, some power function (e.g., $a^2$) to represent non-linear components in costs.

We test various combinations of the features mentioned above and name them Value Function Approximations (VFA) 1, 2 and 3 (see the Appendix for their settings).

## Computational Results

Intuitively, the postponed freights and their characteristics influence the future costs of a decision. However, measuring how these characteristics influence the costs, and thus determining which VFA is the best one to use, is challenging. For small instances of the problem, one option to determine the best set of features to use is to perform a linear regression between the optimal values of all states of the MDP and the basis functions corresponding to each set of features, and choose the set with the highest coefficient of determination $R^2$. Another option, applicable to medium sized instances of the problem is to calculate the average costs of a subset of states, using each set of features, in three steps: (1) run the ADP algorithm for a subset of all states, using the different sets of features, (2) simulate the resulting policies for a number of iterations, and (3) repeat the first and second step a number of replications. In the case of this small example, we perform the two options and present the results in Table 3.3 and Fig. 3.8. For the second option, we simulate the resulting policies of all states and show the average difference between the average costs of the simulation and the optimum value of each state. Although the differences among the sets of features in both tests are small, we note that considering them one at a time would lead to different conclusions. With the coefficient of determination of the linear regression, VFA 2 would be selected as the best. However, with the average costs approach, VFA 3 would be selected. In addition to having the lowest average difference, VFA 3 also has the smallest variance of the three sets of features.

Table 3.3: Performance of the different VFAs

| Test indicator | Lookup-table | VFA 1 | VFA 2 | VFA 3 |
|---|---|---|---|---|
| $R^2$ | – | 0.8897 | 0.8915 | 0.8897 |
| Average difference (%) | 7.50 | 2.67 | 2.45 | 2.36 |

The two aforementioned tests of the various combinations of features consider all states of the MDP. A third approach to decide which basis functions to use, which is applicable to large instances, is to pick some initial state (or multiple initial states) and compare (1) the values learned by the ADP algorithm using various

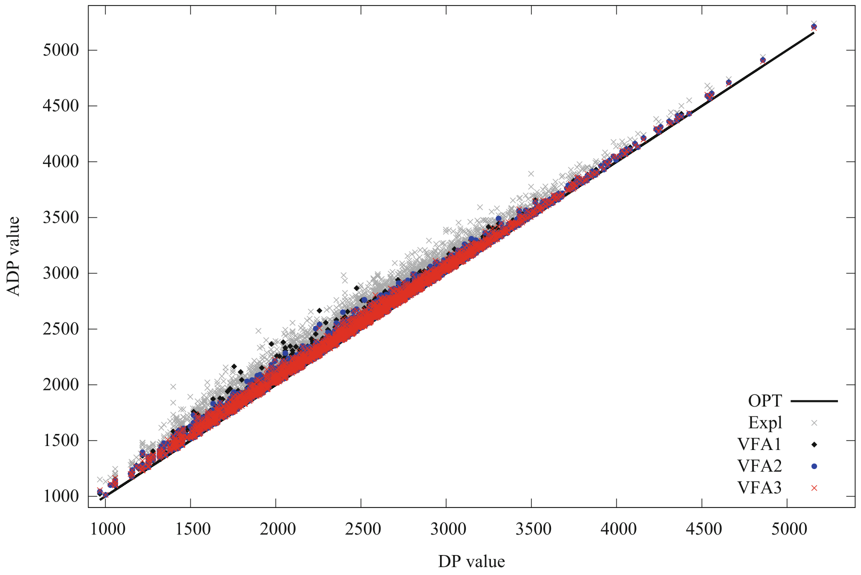


Fig. 3.8: Average performance of the ADP algorithm for the different VFAs compared to the optimal values for all states, using $N = 250$, $M = 250$, $O = 100$, and $K = 10$

sets of features and (2) the resulting performance of the policy resulting from these values. We perform this approach for the two states introduced before, and show the results in Fig. 3.9. Since we use a small problem instance, we also show the optimal value in the figures, as well as the lookup-table approach mentioned earlier in this section (denoted by "Expl"). For all sets of features (VFAs), the ADP algorithm runs for 250 iterations using a double pass approach. In addition to the tests of each VFA, we also test each VFA with an $\varepsilon$-greedy approach ($\varepsilon = 0.05$), and denote these on the graphs by "VFAeps", since this approach yielded good results in our example.

For State 1 in Fig. 3.9, we observe some differences among the VFAs estimated (learned) value and the average costs (performance) of the resulting policy. These differences can lead to choosing different sets of features. On the one hand, the differences among the learned values of the three sets of features indicate that VFA1eps is the best. On the other hand, there are no clear differences among the average costs of all VFAeps, indicating that the three sets perform equally well when using the $\varepsilon$-greedy approach (and all better than all no-$\varepsilon$ VFAs). Furthermore, we observe that the $\varepsilon$-greedy approach improves the estimated values in VFA1 and VFA2 (i.e., VFAeps $\leq$ VFA), but not in VFA3. In the case of the average costs, the $\varepsilon$-greedy approach improves all VFAs in a way that their performance is almost the same.

The results for State 1 can lead to the conclusion that a proper tuning of the exploration/exploitation tradeoff (e.g., via the $\varepsilon$-greedy) can have a larger impact on the performance than the set of features chosen. However, an explanation on why this is the case for this state has to do with the state and the sets of features themselves. State 1 is an almost empty state (i.e., only one freight), which means most basis functions of the three sets we test return zero. Remind that the updating algorithm can only determine how significant the weight of a basis function is as long as it observes it. When only one basis function is observed, and this basis function behaves similarly in all sets of features, the updating algorithm will assign similar weights and thus the resulting policies will be approximately the same.

For State 2 in Fig. 3.9, we observe significant differences among the estimated values of the VFAs, but not among the average costs of the resulting policies. Clearly, VFA2 and VFA3eps have the best estimated value, and VFA3 the worst (even worse than the lookup-table approach). However, when looking at the average costs, the policy from all three VFAs (without the $\varepsilon$-greedy approach) seem to achieve the same costs, between 1 and 2% away from the optimal costs. Moreover, the second-best learning set of features (VFA3eps) is now performing second-worst of all seven value function approximation methods tested. This indicates that having good estimates of the value of states do not necessarily result in a good performance.

When looking at all four figures, we can conclude that deciding on which set of features to use requires careful design and testing, and that the quality of the chosen set of features (basis functions) is heavily problem/state dependent. An explanation on this situation has to do with two characteristics of how the basis functions approximate the future costs. First, all weights of the basis functions, which determine the output policy of the ADP algorithm, can only be updated (i.e., improved)

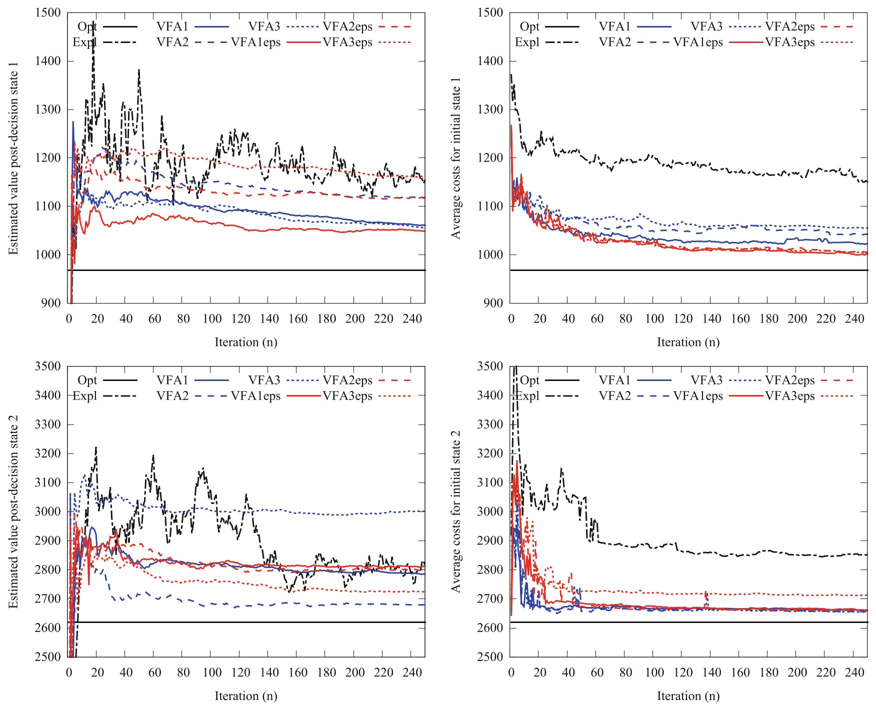


Fig. 3.9: Learned values (*left*) and average cost performance (*right*) of the ADP algorithm for the different VFAs for State 1 (*top*) and State 2 (*bottom*), using $N = 250$, $M = 1$, $O = 100$, and $K = 10$

as long as the basis functions are non-zero. In State 2, which contains many basis functions with a non-zero value, the performance of all VFAs is significantly better than in State 1, which contains mostly basis functions with a value of zero. On average, all six VFAs achieve 2% higher-than-optimal costs in State 2, while they achieve 6% higher-than-optimal costs in State 1. Second, the magnitude with which the weight is updated depends on how much the value of the basis function varies among the different iterations of the ADP algorithm. These might lead to poorly estimating the value itself. In State 2, the difference between the best and worst learning VFA is larger than in State 1. In this example problem, 1.2 freights arrive on average per day (with at most two freights). This means that State 1 is a state one can expect on average whereas State 2 is an exceptionally busy state. Additionally, with the short horizon considered in this problem, the initial conditions (state) can have a large impact on the optimal costs. Thus, problem/state characteristics must be considered when using the basis functions approach.

Besides the need for an evaluation methodology, the observed performance differences between different initial states also gives rise to new VFA designs that use basis functions. For example, using aggregated designs based on categorization of states can prevent basis function values of zero and can reduce the variation among basis function values. Designing such a VFA with the right set of features is both an art and a science. With creativity about potential causes of future costs, as well as their limits within a problem, efficient and accurate designs can be developed. With structured evaluation methodologies (e.g., regression analysis, design of experiment techniques, statistical control methods), these designs can be tested and further improved to tune the ADP algorithm to a specific problem.

For further reading on this problem, we refer to [9]. In addition, we refer to [18] for a similar ADP approach on a completely different transportation problem.

## 3.4 A Healthcare Example

In this third and final example, we repeat the same steps as with the previous two examples, with the difference that we only focus on the modeling part. We omit the experimental results of the MDP and ADP model, for which we refer to [6].

### *3.4.1 Problem Introduction*

The problem concerns tactical planning in a hospital, which involves the allocation of resource capacities and the development of patient admission plans. More concretely, tactical plans distribute a doctor's time (resource capacity) over various activities and control the number of patients that should be treated at each care stage (e.g., surgery). The objective is to achieve equitable access and treatment duration for patients. Each patient needs a set of consecutive care stages, which we denote as a care process. Patients are on a waiting list at each care stage in their care process, and the time spent on this waiting list is called access time. Fluctuations in patient arrivals and resource availabilities result in varying access times for patients at each stage in their care process, and for hospitals, this results in varying resource utilization and service levels. To mitigate and address these variations, tactical planning of hospital resources is required.

The planning horizon is discretized in consecutive time periods $\mathcal{T} = \{1, 2, \ldots, T\}$. We include a set of resource types $\mathcal{R} = \{1, 2, \ldots, R\}$ and a set of patient queues $\mathcal{J} = \{1, 2, \ldots, J\}$. We define $\mathcal{J}^r \subseteq \mathcal{J}$ as the subset of queues that require capacity of resource $r \in \mathcal{R}$. Each queue $j \in \mathcal{J}$ requires a given amount of time units from one or more resources $r \in \mathcal{R}$, given by $s_{j,r}$, and different queues may require the same resource. The number of patients that can be served by resource $r \in \mathcal{R}$ is limited by the available resource capacity $\eta_{r,t}$ in time period $t \in \mathcal{T}$. The resource capacity $\eta_{r,t}$ is given in the same time unit as $s_{j,r}$.

After being treated at a queue $j \in \mathcal{J}$, patients either leave the system or join another queue. To model these transitions, we introduce $q_{j,i}$, which denotes the fraction of patients that will join queue $i \in \mathcal{J}$ after being treated in queue $j \in \mathcal{J}$. To capture arrivals to and exits from outside the "hospital system", we introduce the element 0 (note that the set $\mathcal{J}$ carries no 0-th element by definition). The value $q_{j,0} = 1 - \sum_{i \in \mathcal{J}} q_{j,i}$ denotes the fraction of patients that leave the system after being treated at queue $j \in \mathcal{J}$.

In addition to demand originating from the treatment of patients at other queues within the system, demand may also arrive to a queue from outside the system. The number of patients arriving from outside the system to queue $j \in \mathcal{J}$ at time $t \in \mathcal{T}$ is given by $\lambda_{j,t}$, and the total number of arrivals to the system is given by $\lambda_{0,t}$.

Patients are transferred between the different queues according to transition probabilities $q_{j,i}, \forall j, i \in \mathcal{J}$ independent of their preceding stages, independent of the state of the network and independent of the other patients. Patients arrive at each queue from outside the system according to a Poisson process with rate $\lambda_{j,t}$, $\forall j \in \mathcal{J}, t \in \mathcal{T}$. The external arrival process at each queue $j \in \mathcal{J}$ in time period $t \in \mathcal{T}$ is independent of the external arrival process at other queues and other time periods. Since all arrival processes are independent, we obtain $\lambda_{0,t} = \sum_{j=1}^{J} \lambda_{j,t}$, $\forall t \in \mathcal{T}$. We introduce $\mathcal{U} = \{0, 1, 2, \ldots, U\}$ to represent the set of time periods patients can be waiting, i.e., if we decide not to threat a patient that already waited for $U$ time periods, we assume his/her waiting time remains $U$ time periods.

## 3.4.2 MDP Model

We subsequently present the following elements of the MDP model: the state (Sect. 3.4.2.1), the decision (Sect. 3.4.2.2), the costs (Sect. 3.4.2.3), the new information and transition function (Sect. 3.4.2.4), and the solution (Sect. 3.4.2.5).

### 3.4.2.1 State

We introduce $S_{t,j,u}$ as the number of patients in queue $j \in \mathcal{J}$ at time $t \in \mathcal{T}$ with a waiting time of $u \in \mathcal{U}$. The state of the system at time period $t$ can be written as $S_t = (S_{t,j,u})_{j \in \mathcal{J}, u \in \mathcal{U}}$.

### 3.4.2.2 Decision

The decision $x_{t,j,u}$ is how many patients to treat in queue $j \in \mathcal{J}$ at time $t \in \mathcal{T}$, with a waiting time of $u \in \mathcal{U}$. This decision needs to be made for all queues and waiting times, represented by $x_t = (x_{t,j,u})_{j \in \mathcal{J}, u \in \mathcal{U}}$. The set $\mathcal{X}_t$ of feasible decisions at time $t$ is given by

$$\mathcal{X}_t = \{\, x_t \,|\, \begin{array}{ll} x_{t,i,u} \leq S_{t,i,u}, & \forall i \in \mathcal{J}, t \in \mathcal{T}, u \in \mathcal{U} \\ \sum_{j\in\mathcal{J}^r} s_{j,r} \sum_{u\in\mathcal{U}} x_{t,j,u} \leq \eta_{r,t}, & \forall r \in \mathcal{R}, t \in \mathcal{T} \\ x_{t,j,u} \in \mathbb{Z}_+ & \forall i \in \mathcal{J}, t \in \mathcal{T}, u \in \mathcal{U} \}. \end{array} \tag{3.24}$$

As given in (3.24), the set of feasible decisions in time period $t$ is constrained by the state $S_t$ and the available resource capacity $\eta_{r,t}$ for each resource type $r \in \mathcal{R}$.

#### 3.4.2.3 Costs

The cost function $C_t(S_t, x_t)$ related to our current state $S_t$ and decision $x_t$ is set-up to control the waiting time per stage in the care process, so per individual queue ($j \in \mathcal{J}$). We choose the following cost function, which is based on the number of patients for which we decide to wait at least one time unit longer

$$C_t(S_t, x_t) = \sum_{j\in\mathcal{J}} \sum_{u\in\mathcal{U}} c_{j,u} (S_{t,j,u} - x_{t,j,u}), \qquad \forall t \in \mathcal{T}. \tag{3.25}$$

In general, higher $u \in \mathcal{U}$ will have higher costs as it means a patient has a longer total waiting time.

#### 3.4.2.4 New Information and Transition Function

The vector $W_t$ containing the new information, consists of new patient arrivals and outcomes for transitions between queues. We distinguish between *exogenous* and *endogenous* information in $W_t = \left(\widehat{S}^e_t, \widehat{S}^o_t(x_{t-1})\right)$, $\forall t \in \mathcal{T}$, where the exogenous $\widehat{S}^e_t = \left(\widehat{S}^e_{t,j}\right)_{\forall j \in \mathcal{J}}$ represents the patient arrivals from outside the system, and the endogenous $\widehat{S}^o_t(x_{t-1}) = \left(\widehat{S}^o_{t,j,i}(x_{t-1})\right)_{\forall i,j \in \mathcal{J}}$ represents the patient transitions to other queues as a function of the decision vector $x_{t-1}$. $\widehat{S}^o_{t,j,i}(x_{t-1})$ gives the number of patients transferring from queue $j \in \mathcal{J}$ to queue $i \in \mathcal{J}$ at time $t \in \mathcal{T}$, depending on the decision vector $x_{t-1}$.

We use the following transition function:

$$S_t = S^M(S_{t-1}, x_{t-1}, W_t), \tag{3.26}$$

where

$$S_{t,j,0} = \widehat{S}^e_{t,j} + \sum_{i\in\mathcal{J}} \widehat{S}^o_{t,i,j}(x_{t-1,i}), \quad \forall j \in \mathcal{J}, t \in \mathcal{T}, \tag{3.27}$$

$$S_{t,j,U} = \sum_{u=U-1}^{U} \left(S_{t-1,j,u} - x_{t-1,j,u}\right), \quad \forall j \in \mathcal{J}, t \in \mathcal{T}, \tag{3.28}$$

$$S_{t,j,u} = S_{t-1,j,u-1} - x_{t-1,j,u-1}, \quad \forall j \in \mathcal{J}, t \in \mathcal{T}, u \in \mathcal{U} \setminus \{0, U\}, \tag{3.29}$$

are constraints to ensure that the waiting list variables are consistently calculated. Constraint (3.27) determines the number of patients entering a queue. Constraint (3.28) updates the waiting list for the longest waiting patients per queue. The state $S_{t,j,U}$, for all $t \in \mathcal{T}$ and $j \in \mathcal{J}$, holds all patients that have been waiting $U$ time periods and longer. Constraint (3.29) updates the waiting list variables at each time period for all $u \in \mathcal{U}$ that are not covered by the first two constraints. All arrivals in time period $t \in \mathcal{T}$ to queue $j \in \mathcal{J}$ from outside the system ($\widehat{S}^{e}_{t,j}$) and from internal transitions ($\sum_{i \in \mathcal{J}} \widehat{S}^{o}_{t,i,j}(x_{t-1,i})$) are combined in (3.27).

#### 3.4.2.5 Solution

Again, the formal objective of the model is to find the policy $\pi \in \Pi$ that minimizes the expected costs over the planning horizon, given an initial state $S_0$, as seen in (3.1). The exact DP-problem is restricted by limiting the number of patients that can be waiting in each queue to a given maximum. To illustrate the size of the state space for our problem, suppose that $\hat{M}$ gives the maximum number of patients per queue and per number of time periods waiting. The number of states is then given by $\hat{M}^{(|\mathcal{J}| \cdot |\mathcal{U}|)}$. We can solve (3.4) plugging in the transition function (3.26) and specifying the probability $\mathbb{P}(W_{t+1} = \omega)$, which can be found in [6].

### *3.4.3 Approximate Dynamic Programming*

The ADP approach is presented using a similar setup as used in the previous examples. We subsequently present the post-decision state (Sect. 3.4.3.1), the forward dynamic programming approach (Sect. 3.4.3.2), and the use of value function approximations (Sect. 3.4.3.3).

#### 3.4.3.1 Post-decision State

The post-decision state $S_t^{x,n}$ represents the expected results of the decision $x_t$ taken in state $S_t^n$. More specifically, we subtract the number $x_{t,j,u}$ of patients we decided to treat and use the expected patient transitions $q_{i,j}$ to determine the next location for each of the patients we decided to treat.

The transitions take place as follows. In addition to the transition function (3.26), which gives the transition from the state $S_t^n$ to the state $S_{t+1}^n$, we introduce a transition function $S^{M,x}(S_t^n, x_t)$, which gives the transition from the state $S_t^n$ to the post-decision state $S_t^{x,n}$. This function is given by:

$$S_t^{x,n} = S^{M,x}(S_t^n, x_t), \tag{3.30}$$

with

$$S^{x,n}_{t,j,0} = \sum_{i\in\mathcal{J}} \sum_{u\in\mathcal{U}} q_{i,j} x_{t,i,u} \qquad \forall j \in \mathcal{J}, t \in \mathcal{T} \tag{3.31}$$

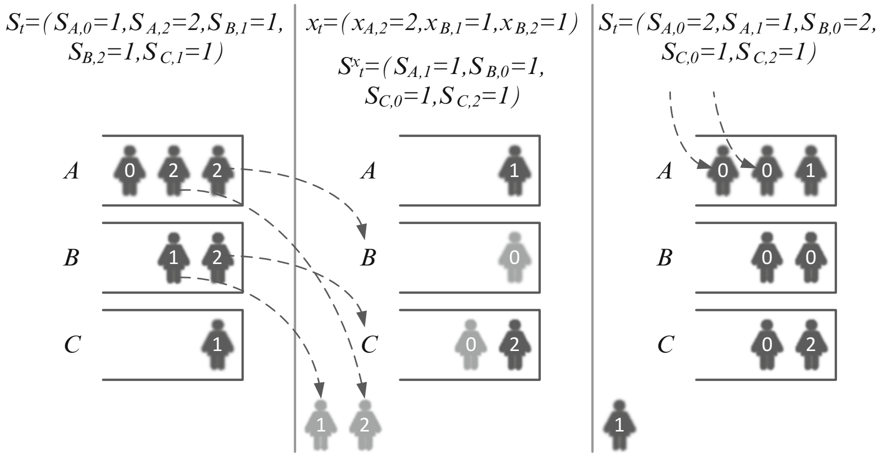
$$S^{x,n}_{t,j,U} = \sum_{u=U-1}^{U} (S_{t,j,u} - x_{t,j,u}) \qquad \forall j \in \mathcal{J}, t \in \mathcal{T} \tag{3.32}$$

$$S^{x,n}_{t,j,u} = S_{t,j,u-1} - x_{t,j,u-1} \qquad \forall j \in \mathcal{J}, t \in \mathcal{T}, u \in \mathcal{U} \setminus \{0, U\}. \tag{3.33}$$

The transition function (3.30) closely resembles (3.26), except that the external arrivals to the system and the final realization of the patient transitions $q_{i,j}$ are not included.

Due to the patient transfer probabilities, the transition function (3.30) may result in non-integer values for the post-decision state. We do not round these values as the post-decision state is only used to provide a value estimate from a particular combination of a state and a decision. Hence, the post-decision state is only used as an 'estimate' of the future state. The post-decision state will not be used to compute the transition from state $S_t$ to state $S_{t+1}$. Within the ADP algorithm, we use the original transition function (3.26) to compute the state in the next time period. As a result, the post-decision state will not cause any state to become non-integer.

The actual realizations of new patient arrivals and patient transitions in a time period will be incorporated in the transition to the state in the next time period. An illustration of the transition of states can be found in Fig. 3.10.



$S_t=(S_{A,0}=1, S_{A,2}=2, S_{B,1}=1, S_{B,2}=1, S_{C,1}=1)$
$x_t=(x_{A,2}=2, x_{B,1}=1, x_{B,2}=1)$
$S^x_t=(S_{A,1}=1, S_{B,0}=1, S_{C,0}=1, S_{C,2}=1)$
$S_t=(S_{A,0}=2, S_{A,1}=1, S_{B,0}=2, S_{C,0}=1, S_{C,2}=1)$
A
B
C

Fig. 3.10: Transition of states in the healthcare problem

### 3.4.3.2 Forward Dynamic Programming

We use a the same forward dynamic programming approach as presented in Sect. 3.3.3.2.

#### 3.4.3.3 Value Function Approximation

Again, the challenge is to design a proper approximation for the 'future' costs $\overline{V}_t^n\left(S_t^{x,n}\right)$ that is computationally tractable and provides a good approximation of the actual values. Similar to the previous example, we make use of basis functions.

For our application, we make the assumption that the properties of each queue are independent from the properties of the other queues, so that we can define basis functions for each individual queue that describe important properties of that queue. For the basis functions, we choose to use the features 'The number of patients in queue $j$ that are waiting for $u$ periods'. These features result in the following basis functions that will be used in the ADP algorithm: $S_{t,j,u}, \forall j \in \mathcal{J}, \forall u \in \mathcal{U}, t = 1$. The basis functions explain a large part of the variance in the computed values with the exact DP approach ($R^2 = 0.954$) and they can be straightforwardly obtained from the post decision state. The weights $\theta^n$ in the value function approximations are initialized to $\theta^0 = 1$ for all time periods, and we use the matrix $B^0 = \varepsilon I$ as explained before. We use the double pass version of the ADP algorithm and determine the stepsize $\alpha$ using nonstationary least squares with $\delta = 0.99$. All other settings can be found in [6].

In case there is no independent constant in the set of predictors $\mathcal{F}$ in a linear regression model, the model is forced to go through the origin (all dependent and independent variables should be zero at that point). This may cause a bias in the predictors. To prevent this bias, we add a constant term as one of the elements in $\mathcal{F}$. The feature weight $\theta_f^n$ may vary, but the feature value $\phi_f\left(S_t^{x,n}\right)$ of this constant is always 1, independent of the state $S_t^{x,n}$.

We have calculated the ADP-algorithm for 5000 random states and found that the values found with the ADP algorithm and the value from the exact DP solution converge. For these 5000 random states, there is an average deviation between the value approximated with the ADP algorithm and the value calculated with the exact DP approach of 2.51%, with a standard deviation of 2.90%, after 500 iterations. This means the ADP algorithm finds slightly larger values on average than the exact DP approach. This may be caused by the truncated state space, as explained before. The calculation time of the ADP algorithm is significantly lower than the calculation of the exact DP solution. Obtaining the DP solution requires over 120 h. Calculating the ADP solution for a given initial state (with $N = 500$) takes on average only 0.439 s. For a complete analysis of this approach, we refer to [6].

## 3.5 What's More

ADP is a versatile framework that is studied and applied to a growing number of diverse problems. Naturally, diverse problems require a deeper focus on diverse aspects of ADP. In some problems, a correct design of a value function is of most importance for approximating the optimal solution of an MDP, whereas in others, the right tuning of exploration vs exploitation parameters has a higher impact.

Furthermore, in some practical applications, approximating a restricted policy might be better than approximating the values. In this section, we briefly touch upon some of these aspects. We subsequently present various options for policies (Sect. 3.5.1), value function approximations (Sect. 3.5.2), and handling the exploration vs exploitation tradeoff (Sect. 3.5.3).

### *3.5.1 Policies*

In this chapter, we used policies based on value function approximations. There are many other options, like the use of myopic policies, look-ahead policies (rolling horizon procedures that optimize over multiple time periods into the future), and policy function approximations (analytic functions that return an action for each state). And, of course it is possible to use hybrids.

The approach used in this chapter relies on approximate value iteration. Another option is approximate policy iteration. In this strategy, we simulate a policy a number of 'inner' iterations over some horizon. During these inner iterations, we fix the policy (typically by fixing the value function approximation) to obtain a better estimate of the value of being in a state. We refer to [10, Chaps. 9 and 10] for more information on this. Finally, besides value iteration and policy iteration, we can also use the linear programming method. This method—that for MDPs suffers from the curse of dimensionality since we need a decision variable for each state, and a constraint for each state-action pair—receives attention in the ADP community due to [2], where ADP concepts are applied to this method, incorporating value function approximations into the linear program and sampling of the constraints.

More information on different types of policies, and ADP modeling in general, can be found in [11–13] and, using examples from transportation and logistics, [14]. In these works, also the relationship between (approximate) dynamic programming and other techniques as stochastic programming, simulation, and stochastic search, is discussed. A comparison of different ADP techniques, using an energy storage problem, is given in [7].

### *3.5.2 Value Function Approximations*

Value function approximations can be divided into lookup tables (including state space aggregation, hierarchical aggregation, and representatives), parametric models (basis functions, piece-wise linear functions, and neural networks), and nonparametric models (kernel regression and support vector machines). In general, approximating value functions involves the application of statistical methods to estimate the value of being in a state. One specific technique, applicable to the approaches considered in this chapter, involves the selection of basis functions, see [17]. For an overview of statistical learning techniques that can be used in this setting, we refer to [5].

The estimation of value function approximations, however, involves much more than the application of statistical methods. A unique setting of ADP is that value function approximations have to be estimated recursively. Especially during the initial iterations, our decisions are influenced by the initialized values and might be strongly biased by the number of measurements taken from the different states. In addition, testing the performance of a VFA design is a task that requires diverse methods as well, as we illustrated in this chapter.

### *3.5.3 Exploration vs Exploitation*

The exploration vs exploitation tradeoff involves the decisions whether to explore states just to learn their value or to visit the states that appear to be the best. For this purpose, we introduced the $\varepsilon$-greedy policy. The disadvantage of this policy is that the learning rate remains constant and there is no focus on certain areas of the state space. A 'good' policy supports the balance between the estimated value of states and the uncertainty about these values. This problem received considerable attention by the machine learning community (problems related to the Multi-armed Bandit Problem, Ranking and Selection, Bayesian Global Optimization, etc.). These techniques can also be used within ADP. An example can be found in [15], where a new exploration strategy is proposed based on the knowledge gradient concept, in which the uncertainty about the value function is explicitly expressed using a Bayesian model with correlated beliefs. The hierarchical aggregation method described in this chapter has also been extended with a Bayesian belief model in [8]. We incorporated this Hierarchical Knowledge Gradient method within ADP, using a similar approach as presented in [15], and tested it on the nomadic trucker problem from Sect. 3.2. For all iterations $10 \leq n \leq 250$, this exploration technique consistently outperforms the other policies show in Fig. 3.3, both with respect to the learned value functions and the resulting performance. For a more in-depth discussion of strategies to balance exploration and exploitation, we refer to [16], [10, Chap. 10], and [13].

## Appendix

### *Nomadic Trucker Settings*

Transportation takes place in a square area of $1000 \times 1000$ miles. The locations lie on a $16 \times 16$ Euclidean grid placed on this area, where each location $i \in \mathcal{L}$ is described by an $(x_i, y_i)$-coordinate. The first location has coordinate $(0,0)$ and the last location (location 256) has coordinate $(1000,1000)$. The minimum distance between two locations is $1000/15$.

For each location $i \in \mathcal{L}$, there is a number $0 \leq b_i \leq 1$ representing the probability that a load originating at location $i$ will appear at a given time step. The probability that, on a given day of the week $d$, a load from $i$ to $j$ will appear is given by $p^d_{ij} = p_d b_i (1 - b_j)$, where $p_d$ gives the probability of loads appearing on a given day of the week $d$. The origin probabilities $b_i$ are given by

$$b_i = \rho \left( 1 - \frac{f(x_i, y_i) - f^{min}}{f^{max} - f^{min}} \right), \tag{3.34}$$

where $\rho$ gives the arrival intensity of loads, and $f(x_i, y_i)$ is the Six-hump camel back function given by $f(x_i, y_i) = 4x_i^2 - 2.1x_i^4 + \frac{1}{3}x_i^6 + x_i y_i - 4y_i^2 + 4y_i^4$ on the domain $(x_i, y_i) \in [-1.5, 2] \times [-1, 1]$. The highest value is achieved at coordinate $(2, 1)$, with a value of $\approx 5.73$, which we reduce to 5 to create a somewhat smoother function (still the second highest value is $\approx 4.72$). Next, the values $f(x_i, y_i)$ are scaled to the domain $(x_i, y_i) \in [0, 0] \times [1000, 1000]$. The values $f^{min} = min_{i \in \mathcal{L}} f(x_i, y_i) \approx -1.03$ and $f^{max} = max_{i \in \mathcal{L}} f(x_i, y_i) = 5$ are used to scale $f(x_i, y_i)$ between $[0, 1]$. An impression of the resulting origin probabilities $B_i$ is given in Fig. 3.11.

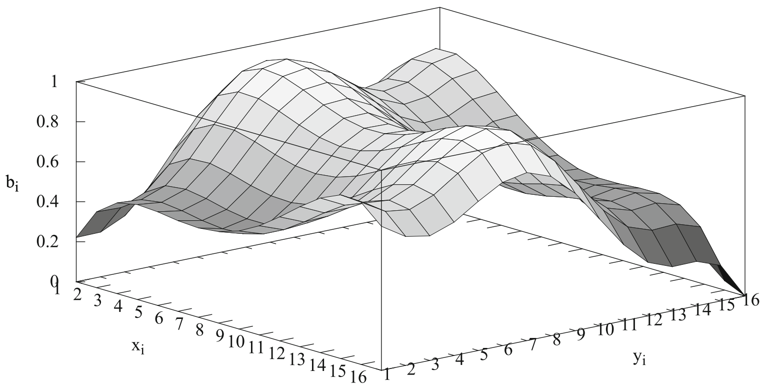


Fig. 3.11: Origin probabilities for the 256 locations

We set $\rho = 1$, which corresponds with an expectation of approximately 93.14 outgoing loads from the most popular origin location on the busiest day of the week. We use a load probability distribution $p^d = (1, 0.8, 0.6, 0.7, 0.9, 0.2, 0.1)$, for $d$ from Monday till Sunday, which represents the situation in which loads are more likely to appear during the beginning of the week (Mondays) and towards the end (Fridays).

The results for the infinite horizon multi-attribute version of the nomadic trucker problem can be found below (Fig. 3.12).

## *Freight Consolidation Settings*

Either one or two freights arrive each period (i.e., $\mathcal{F} = \{1, 2\}$), with probability $p^F_f = (0.8, 0.2)$ for $f \in \mathcal{F}$. Each freight that arrives has destination $d \in \mathcal{D} = \{1, 2, 3\}$

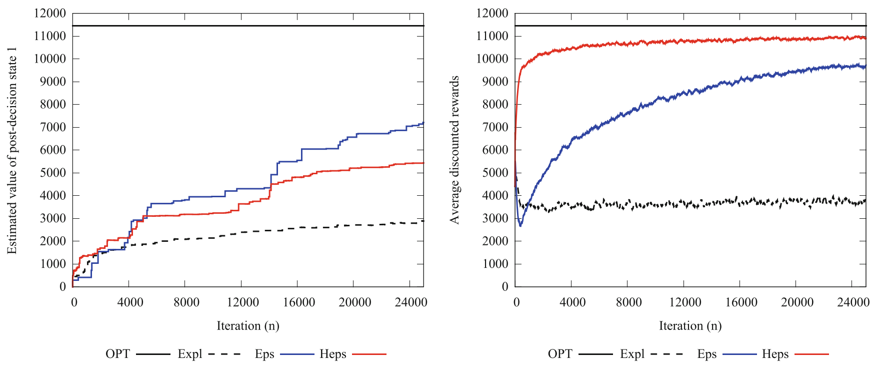


Fig. 3.12: Infinite horizon multi-attribute case: resulting estimate $\overline{V}_0^n\left(S_0^{x,n}\right)$ (*left*) and realized rewards (*right*), using $N = 25{,}000$, $M = 10$, $O = 1000$ and $K = 10$. For the rewards resulting from the simulations, the 2500 observations are smoothed using a window of 10. For the policies Expl and Eps, the BAKF stepsize is used

with probability $p_d^D = (0.1, 0.8, 0.1)$, is already released for transportation (i.e., $r \in \mathcal{R} = \{0\}$ and $p_r^R = 1$), and has time-window length $k \in \mathcal{K} = \{0, 1, 2\}$ with probability $p_k^K = (0.2, 0.3, 0.5)$.

The costs are defined as follows. The long-haul, high capacity vehicle costs (per subset of destinations visited) are $C_{\mathcal{D}'} = (250, 350, 450, 900, 600, 700, 1000)$ for $\mathcal{D}' = (\{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\})$, respectively. These costs are for the entire long-haul vehicle, independent on the number of freight consolidated. Furthermore, we consider there are no costs for the long-haul vehicle if no freights are consolidated. The alternative, low capacity mode costs (per freight) are $B_d = (500, 1000, 700)$ for $d \in \mathcal{D}$. There is no discount factor, i.e. $\gamma = 1$.

We build three different sets of features based on a common "job" description used in transportation settings: *MustGo*, *MayGo*, and *Future* freights. MustGo freights are those released freights whose due-day is immediate. MayGo freights are those released freights whose due-day is not immediate. Future freights are those that have not yet been released. We use the MustGo, MayGo and Future adjectives in destinations as well, with an analogous meaning to those of freight. In Table 3.4 we show the three sets of features, which we name Value Function Approximation (VFA) 1, 2, and 3. All feature types in this table are related to the freights of a post-decision state. The symbol '*' denotes a VFA set containing a feature type. All feature types are numerical, and either indicate (i.e., 1 if yes, 0 if no), count (1,2,...), number (add), or multiply (i.e., product between two numbers) the different type of freights and destinations. Between parentheses we show the number of basis functions (i.e., independent variables) that a feature type has for the test instance. For example, there is one post-decision state variable per destination, per time-window length, thus all post-decision state variables are $3 * 3 = 9$. The constant feature equals one for all post-decision states, and the weights $\theta_a^n$ are all initialized with one.

Table 3.4: Various sets of features (basis functions of a post-decision state)

| Feature type | VFA 1 | VFA 2 | VFA 3 |
|---|---|---|---|
| All post-decision state variables (9) | * | * | * |
| All post-decision state variables squared (9) | * | – | – |
| Count of MustGo destinations (1) | * | * | * |
| Number of MustGo freights (1) | * | * | * |
| Product of MustGo destinations and MustGo freights (1) | * | – | – |
| Count of MayGo destinations (1) | * | * | * |
| Number of MayGo freights (1) | * | * | * |
| Product of MayGo destinations and MayGo freights (1) | * | – | – |
| Count of Future destinations (1) | * | * | * |
| Number of Future freights (1) | * | * | * |
| Product of Future destinations and Future freights (1) | * | – | – |
| Indicator MustGo freights per destination (3) | – | * | – |
| Indicator MayGo freights per destination (3) | – | * | – |
| Indicator Future freights per destination (3) | – | * | – |
| Number of all freights (1) | * | * | * |
| Constant (1) | * | * | * |

## References

1. R. Bellman, *Dynamic Programming*, 1st edn. (Princeton University Press, Princeton, NJ, 1957)
2. D.P.D. Farias, B.V. Roy, On constraint sampling in the linear programming approach to approximate dynamic programming. Math. Oper. Res. **29**(3), 462–478 (2004)
3. A.P. George, W.B. Powell, Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. Mach. Learn. **65**(1), 167–198 (2006)
4. A.P. George, W.B. Powell, S.R. Kulkarni, S. Mahadevan, Value function approximation using multiple aggregation for multiattribute resource management. J. Mach. Learn. Res. **9**, 2079–2111 (2008)
5. T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*. Springer Series in Statistics (Springer, New York, NY, 2001)
6. P.J.H. Hulshof, M.R.K. Mes, R.J. Boucherie, E.W. Hans, Patient admission planning using approximate dynamic programming. Flex. Serv. Manuf. J. **28**(1), 30–61 (2016)
7. D.R. Jiang, T.V. Pham, W.B. Powell, D.F. Salas, W.R. Scott, A comparison of approximate dynamic programming techniques on benchmark energy storage problems: does anything work?, in *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 2014, pp. 1–8
8. M.R.K. Mes, W.B. Powell, P.I. Frazier, Hierarchical knowledge gradient for sequential sampling. J. Mach. Learn. Res. **12**, 2931–2974 (2011)

9. A. Pérez Rivera, M.R.K. Mes, Dynamic multi-period freight consolidation, in *Computational Logistics*, ed. by F. Corman, S. Voß, R.R. Negenborn. Lecture Notes in Computer Science, vol. 9335 (Springer, Cham, 2015), pp. 370–385
10. W.B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley Series in Probability and Statistics (Wiley, London, 2011)
11. W.B. Powell, Perspectives of approximate dynamic programming. Ann. Oper. Res. **241**(1), 319–356 (2012)
12. W.B. Powell, Clearing the jungle of stochastic optimization, in *Informs Tutorials in Operations Research*, chap. 4 (INFORMS, Hanover, MD, 2014), pp. 109–137
13. W.B. Powell, I.O. Ryzhov, *Optimal Learning and Approximate Dynamic Programming* (Wiley, London, 2013), pp. 410–431
14. W.B. Powell, H.P. Simao, B. Bouzaiene-Ayari, Approximate dynamic programming in transportation and logistics: a unified framework. EURO J. Transp. Logist. **1**(3), 237–284 (2012)
15. I.O. Ryzhov, W.B. Powell, Approximate dynamic programming with correlated bayesian beliefs, in *Proceedings of the 48th Allerton Conference on Communication, Control and Computing* (2010)
16. R.S. Sutton, A.G. Barto, *Introduction to Reinforcement Learning*, 1st edn. (MIT Press, Cambridge, MA, 1998)
17. J.N. Tsitsiklis, B. Roy, Feature-based methods for large scale dynamic programming. Mach. Learn. **22**(1), 59–94 (1996)
18. W. van Heeswijk, M.R.K. Mes, M. Schutten, An approximate dynamic programming approach to urban freight distribution with batch arrivals, in *Computational Logistics*, ed. by F. Corman, S. Voß, R.R. Negenborn. Lecture Notes in Computer Science, vol. 9335 (Springer, Cham, 2015), pp. 61–75