Flex Serv Manuf J (2016) 28:30–61
DOI 10.1007/s10696-015-9219-1

Patient admission planning using Approximate Dynamic Programming

Peter J. H. Hulshof[1] · **Martijn R. K. Mes**[1] · **Richard J. Boucherie**[1] · **Erwin W. Hans**[1]

Published online: 18 April 2015

Abstract Tactical planning in hospitals involves elective patient admission planning and the allocation of hospital resource capacities. We propose a method to develop a tactical resource allocation and patient admission plan that takes stochastic elements into consideration, thereby providing robust plans. Our method is developed in an Approximate Dynamic Programming (ADP) framework and copes with multiple resources, multiple time periods and multiple patient groups with uncertain treatment paths and an uncertain number of arrivals in each time period. As such, the method enables integrated decision making for a network of hospital departments and resources. Computational results indicate that the ADP approach provides an accurate approximation of the value functions, and that it is suitable for large problem instances at hospitals, in which the ADP approach performs significantly better than two other heuristic approaches. Our ADP algorithm is generic, as various cost functions and basis functions can be used in various hospital settings.

Keywords Healthcare · Tactical planning · Resource capacity planning · Patient admission planning · Approximate Dynamic Programming (ADP)

✉ Peter J. H. Hulshof
p.j.h.hulshof@utwente.nl

✉ Martijn R. K. Mes
m.r.k.mes@utwente.nl

[1] Center for Healthcare Operations Improvement and Research (CHOIR), University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

1 Introduction

This paper concerns tactical planning in a hospital setting, which involves the allocation of resource capacities and the development of patient admission plans (Hulshof et al. 2012). 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). One of the main objectives of tactical planning in healthcare is to achieve equitable access and treatment duration for patients (Hulshof et al. 2013).

In tactical planning, the term care process is used for a set of consecutive care stages followed by patients through a hospital. This is the complete path of a patient group through the hospital, such as for example a visit to an outpatient clinic, a visit to an X-ray, and a revisit to the outpatient clinic. Patients are on a waiting list at each care stage in their care process, and the time spent on this waiting list is termed access time. If access times are controlled, this contributes to the quality of care for the patient. The term care process is not to be confused with "clinical pathways", which is described by Every et al. (2000) as "management plans that display goals for patients and provide the sequence and timing of actions necessary to achieve these goals with optimal efficiency". As care processes are defined by multiple steps that link departments and resources together in an integrated network, fluctuations in patient arrivals and resource capacity availability at a single department or resource may affect other departments and resources in the network. For patients, this results in varying access times for each stage in a care process, and for hospitals, this results in varying resource utilizations and service levels. To mitigate and address these variations, re-allocation of hospital resources, incorporating the perspective of the entire care chain (Cardoen and Demeulemeester 2008; Hall 2006; Porter and Teisberg 2007), seems necessary (Hulshof et al. 2013).

The tactical planning problem in healthcare is stochastic in nature. Randomness exists in for example the number of (emergency) patient arrivals and the number of patient transitions after being treated at a particular stage of their care process. Several papers have focused on tactical planning problems that span multiple departments and resources in healthcare (Garg et al. 2010; Kapadia et al. 1985; Nunes et al. 2009) and other industries (Graves 1986). Hulshof et al. (2012, 2013) reviewed the literature and conclude that the available approaches for tactical planning are myopic, are developed to establish longer term cyclical tactical plans, or cannot provide tactical planning solutions for practical, large-sized instances. The authors develop a deterministic method for tactical planning over multiple departments and resources within a mathematical programming framework.

In this paper, we develop a stochastic approach for the tactical planning problem in healthcare by modeling it as a Dynamic Programming problem (DP). Due to the properties of the tactical planning problem, with discrete time periods and transitions that depend on the decision being made, DP is a suitable modeling approach. As problem sizes increase, solving a DP is typically intractable due to the 'curse of dimensionality'. To overcome this problem, an alternative solution approach for real-life sized instances of the tactical planning problem is needed. The field of Approximate Dynamic Programming (ADP) provides a suitable framework to develop such an alternative approach, and we use this framework to develop an innovative solution approach. ADP uses approximations, simulations and decompositions to reduce the dimensions of a large problem, thereby significantly reducing the required calculation time. A comprehensive explanation and overview of the various techniques within the ADP framework are given in Powell (2011). The application of ADP is relatively new in healthcare, it has been used in ambulance planning (Maxwell et al. 2010; Schmid 2012) and patient scheduling (Patrick et al. 2008). Other applications in a wider spectrum of industries include resource capacity planning (Erdelyi and Topaloglu 2010; Schütz and Kolisch 2012), inventory control (Simao and Powell 2009), and transportation (Topaloglu and Powell 2006).

With this paper and the proposed model, we aim to contribute to the existing literature in two ways. First, we develop a theoretical contribution to tactical resource and admission planning in healthcare in the field of Operations Research and Management Science (OR/MS). We develop an approach to develop tactical plans that take randomness in patient arrivals and patient transitions to other stages into account. These plans are developed for multiple resources and multiple patient groups with various care processes, and integrate decision making for a network of hospital departments and resources. The model is designed with a finite horizon, which allows all input to be time dependent. This enables us to incorporate anticipated or forecasted fluctuations between time periods in patient arrivals (e.g., due to seasonality) and resource capacities (e.g., due to vacation or conference visits) in developing the tactical plans. The model can also be used in 'realtime'. If during actual implementation of the tactical plan, deviations from forecasts make reallocation of resource capacity necessary, the developed model can be used to determine an adjusted tactical plan. The model can be extended to include different cost structures, constraints, and additional stochastic elements. Second, the solution approach is innovative as it combines various methods and techniques within the ADP-framework and the field of mathematical programming. Also, the application of ADP is new in tactical resource capacity and patient admission planning, and relatively new in healthcare in general, where it has mainly been applied in ambulance planning (Maxwell et al. 2010; Schmid 2012) and patient scheduling (Patrick et al. 2008).

The main contribution of this paper is a methodology to create real-life tactical plans that takes randomness into account. In addition, we use an innovative combination of methods and techniques within the ADP-framework and the field of mathematical programming. This combination of techniques is relatively new in the area of healthcare. This paper is organized as follows. Section 2 discusses the mathematical problem formulation and describes the exact Dynamic Programming (DP) solution approach for small instances. Section 3 introduces the ADP approaches necessary to develop tactical plans for real-life sized instances. Section 4 discusses computational results and Sect. 5 describes how the model can be used to develop or adjust tactical plans in healthcare. Section 6 concludes this paper.

2 Problem formulation

This section introduces the reader to the problem, the notation, and the patient dynamics in care processes. The tactical planning problem formulation introduced in Hulshof et al. (2013) is extended in this section to include stochastic aspects, such as random patient arrivals and patient transitions between queues.

The planning horizon is discretized in consecutive time periods $\mathcal{T} = \{1, 2, \ldots, T\}$. This finite horizon allows all input to the model to be time dependent and enables incorporating anticipated or forecasted fluctuations between time periods in patient arrivals and resource capacities. We include a set of resource types $\mathcal{R} = \{1, 2, \ldots, R\}$ and a set of patient queues $\mathcal{J} = \{1, 2, \ldots, J\}$ and $\mathcal{J}^r$ as the set of queues that require capacity of resource $r \in \mathcal{R}$.

Each queue $j \in \mathcal{J}$ requires a given amount of (time) capacity from one or more resources, given by $s_{j,r}$, $r \in \mathcal{R}$, 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 $\eta_{r,t}$ uses the same capacity unit as $s_{j,r}$. The service time and resource capacity parameters are primarily used to take the resource constraints into account when planning. In our model, we assume that if service for a patient at queue $j \in \mathcal{J}$ starts in time period t, that service will be finalized before time period $t + 1$.

After being treated at a queue $j \in \mathcal{J}$ at time period $t \in \mathcal{T}$, patients either leave the system immediately or join another queue at time period $t + 1$. 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 $j \in \mathcal{J}$. Our modeling framework allows for different types of transitions, e.g., transitions to any prior or future stage in the same care process, and transitions between queues of different care processes. 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 $\lambda_{0,t}$.

Within the definition of $q_{j,i}$ lies the major assumption of our model:

Assumption 1 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.

For practical purposes in which Assumption 1 does not hold, we can adjust the various care processes to ensure it does hold. For example, if after some stage within a care process, the remainder of the patient's path depends on the current stage, we create a new care process for the remaining stages and patients flow with a certain probability to the first queue in that new care process.

For the arrival processes, we assume the following.

Assumption 2 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.

Given that the arrival processes to the different queues are independent (see Assumption 2), the total number of arrivals to the system follow a Poisson process with rate $\lambda_{0,t} = \sum_{j=1}^{J} \lambda_{j,t}$, $\forall t \in \mathcal{T}$.

We introduce $\mathcal{U} = \{0, 1, 2, \ldots, U\}$ to represent the set of time periods patients can be waiting. To ensure a finite state space, we introduce a bound U on the number of time periods that patients can be waiting. For patients with a waiting time of U time periods at time $t \in \mathcal{T}$, that are not treated at time period t, we keep classifying them as having a waiting time of U time periods. In other words, the bound U represents a waiting time of at least U time periods in our model.

Given Assumption 1, patients are characterized by the queue in which they are waiting and the amount of time they have been waiting at this queue. We introduce

$$S_{t,j,u} = \text{Number of patients in queue j} \in \mathcal{J} \text{at time t} \in \mathcal{T} \text{ with a waiting time of u} \in \mathcal{U}.$$

The state of the system at time period t can be written as $S_t = \left(S_{t,j,u}\right)_{j \in \mathcal{J}, u \in \mathcal{U}}$. We define decisions as actions that can change the state of the system. The decisions are given by

$$x_{t,j,u} = \text{Number of patients to treat in queue j} \in \mathcal{J} \text{at time t} \in \mathcal{T}, \text{with a waiting time of u} \in \mathcal{U}.$$

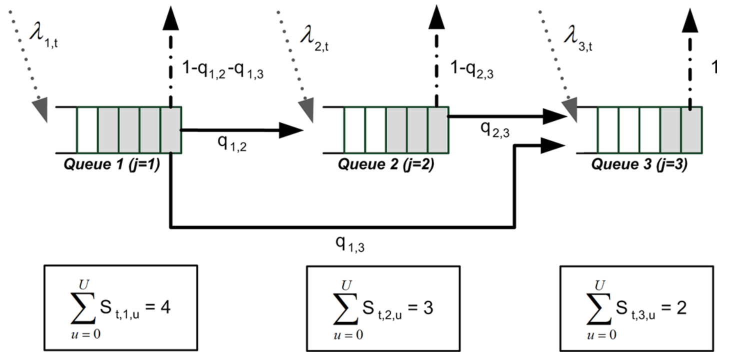

Fig. 1 An illustration of the network dynamics for an example with three queues at time period $t \in \mathcal{T}$. Queue 1 currently has four patients, queue 2 has three patients, and queue 3 has two patients. When a patient is served at queue 1, the patient flows to queue 2 according to probability $q_{1,2}$ and to queue 3 with probability $q_{1,3}$. New patients arrive to queue 1 at a rate $\lambda_{1,t}$

The decision at time period t can be written as $x_t = \left(x_{t,j,u}\right)_{j \in \mathcal{J}, u \in \mathcal{U}}$. The network of queues that is explained above and some key notation is captured in a simplified illustration in Fig. 1.

The cost function $C_t(S_t, x_t)$ related to our current state S_t and decision x_t can be modeled in various ways. The main objectives of tactical planning are *to achieve equitable access and treatment duration for patient groups* and *to serve the strategically agreed number of patients* (Hulshof et al. 2013). The focus in developing this model is on the patient's waiting time (equitable access and treatment duration), and we assume that the strategically agreed number of patients is set in accordance with patient demand (as the model accepts all patients that arrive). The cost function in our model is set-up to control the waiting time per stage in the care process, so per individual queue ($j \in \mathcal{J}$). It is also possible to adapt the cost function for other tactical planning settings, for example to control the total waiting time per individual care process or for all queues that use a particular resource $r \in \mathcal{R}$. We choose the following cost function, which is based on the number of patients for which we decide to wait at least one time period longer

$$C_t(S_t, x_t) = \sum_{j \in \mathcal{J}} \sum_{u \in \mathcal{U}} c_{j,u}\left(S_{t,j,u} - x_{t,j,u}\right), \qquad \forall t \in \mathcal{T}. \tag{1}$$

The cost component $c_{j,u}$ in (1) can be set by the hospital to distinguish between queues $j \in J$ and waiting times $u \in \mathcal{U}$. In general, higher $u \in \mathcal{U}$ will have higher costs as it means a patient has a longer total waiting time. This could be modeled in various ways, for example the cost $c_{j,u}$ could be incrementally, linearly increasing with $u \in \mathcal{U}$. A different perspective is to significantly increase waiting costs after some hospital-specific threshold on the waiting time. The total costs resulting from the model cannot be directly translated into waiting times or monetary costs. Instead, they are merely used to rank one solution over another, to identify which patients from which queues should be served next. As such, the costs $c_{j,u}$ can be used to distinguish or prioritize between queues, but also between waiting times within queues. For example, two time periods waiting for an urgent and important treatment might receive higher 'costs' than eight time periods waiting for a routine check-up. The setting of the costs $c_{j,u}$ by hospital management will certainly involve trial and error in practice. As the purpose of this paper is a theoretical contribution of an ADP algorithm that can be used for tactical planning in hospitals, we will not test the setting of these costs in this paper.

The different forms of randomness that are apparent in the actual system, such as random patient arrivals and uncertainty in patient transitions to other queues, can be incorporated by using the introduced $\lambda_{i,t}$ and $q_{j,i}$, $i,j \in \mathcal{J}, t \in \mathcal{T}$ as parameters for the stochastic processes. To capture all sources of random information, we introduce

W_t = The vector of random variables representing all the new information that becomes available between time t − 1 and t.

The vector W_t contains all the new information, which consists of new patient arrivals and outcomes for transitions between queues. We distinguish between *exogeneous* and *endogeneous* information in

$$W_t = \left(\widehat{S}^e_t, \widehat{S}^o_t(x_{t-1})\right), \qquad \forall t \in \mathcal{T},$$

where the exogeneous $\widehat{S}^e_t = \left(\widehat{S}^e_{t,j}\right)_{\forall j \in \mathcal{J}}$ represents the patient arrivals from outside the system, and the endogeneous $\widehat{S}^o_t(x_{t-1}) = \left(\widehat{S}^o_{t,j,i}(x_{t-1})\right)_{\forall i,j \in \mathcal{J}}$ 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}.

Assumptions 1 and 2 imply that the probability distribution (conditional on the decision) of future states only depends on the current state, and is independent of preceding states in preceding time periods. This means that the described process has the Markov property. We use this property in defining a transition function, S^M, to capture the evolution of the system over time as a result of the decisions and the random information.

$$S_t = S^M(S_{t-1}, x_{t-1}, W_t), \tag{2}$$

where

$$S_{t,j,0} = \widehat{S}^e_{t,j} + \sum_{i \in \mathcal{J}} \widehat{S}^o_{t,i,j}\left(x_{t-1,i}\right), \quad \forall j \in \mathcal{J}, t \in \mathcal{T}, \tag{3}$$

$$S_{t,j,U} = \sum_{u=U-1}^{U} \left(S_{t-1,j,u} - x_{t-1,j,u}\right), \quad \forall j \in \mathcal{J}, t \in \mathcal{T}, \tag{4}$$

$$S_{t,j,u} = S_{t-1,j,u-1} - x_{t-1,j,u-1}, \quad \forall j \in \mathcal{J}, t \in \mathcal{T}, u \in \mathcal{U} \backslash \{0, U\}, \tag{5}$$

are constraints to ensure that the waiting list variables are consistently calculated. Constraint (3) determines the number of patients entering a queue. Constraint (4) 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 (5) updates the waiting list variables at each time period for all $u \in \mathcal{U}$ that are not covered by the first two constraints. The stochastic information is captured in (3). 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 ($\sum_{i \in \mathcal{J}} \widehat{S}^o_{t,i,j}\left(x_{t-1,i}\right)$) are combined in (3).

We aim to find a policy (a decision function) to make decisions about the number of patients to serve at each queue. We represent the decision function by

$$X^\pi_t(S_t) = \text{A function that returns a decision } \mathrm{x_t} \in \mathcal{X}_\mathrm{t}(\mathrm{S_t}), \text{under the policy } \pi \in \Pi.$$

The set Π refers to the set of potential decision functions or policies. $\mathcal{X}_t$ denotes the set of feasible decisions at time t, which is given by

$$\mathcal{X}_t(S_t) = \begin{array}{ll} \{x_t| & \\ x_{t,i,u} \leq S_{t,i,u}, & \forall i \in \mathcal{J}, t \in \mathcal{T}, u \in \mathcal{U} \\ \sum_{j \in \mathcal{J}^r} s_{j,r} \sum_{u \in \mathcal{U}} x_{t,j,u} \leq \eta_{r,t}, & \forall r \in \mathcal{R}, t \in \mathcal{T} \\ x_{t,j,u} \in \mathbb{Z}_+ & \forall i \in \mathcal{J}, t \in \mathcal{T}, u \in \mathcal{U}\} \end{array}. \tag{6}$$

As given in (6), the set of feasible decisions in time period t is constrained by the state space S_t and the available resource capacity $\eta_{r,t}$ for each resource type $r \in \mathcal{R}$. Our goal is to find a policy π, among the set of policies Π, that minimizes the expected costs over all time periods given initial state S_0. This goal is given in

$$\min_{\pi \in \Pi} \mathbb{E}_{W_t \forall t \in \mathcal{T}} \left\{ \sum_{t \in \mathcal{T}} C_t\left(S_t, X_t^\pi(S_t)\right) | S_0 \right\}, \tag{7}$$

where $S_{t+1} = S^M(S_t, x_t, W_{t+1})$. The challenge is to find the best policy $X_t^\pi(S_t)$.

By the principal of optimality (Bellman 1957), we can find the optimal policy by solving

$$V_t(S_t) = \min_{x_t \in \mathcal{X}_t(S_t)} \left(C_t(S_t, x_t) + \mathbb{E}\{V_{t+1}(S_{t+1}) | S_t, x_t, W_{t+1}\}\right), \tag{8}$$

where $S_{t+1} = S^M(S_t, x_t, W_{t+1})$ gives the state S_{t+1} as a function of the current state S_t, the decisions x_t, and the new information W_{t+1}.

To specify the expectation in (8), we introduce a vector w^a consisting of elements w_j^a representing the number of patients arriving at queue j, and use $P(w^a|x_t)$ to denote the probability of w^a given decision x_t. Enumerating the product of the probability and value associated with all possible outcomes of w^a, establishes the expectation in (8)

$$V_t(S_t) = \min_{x_t \in \mathcal{X}(S_t)} \left(C_t(S_t, x_t) + \sum_{w^a} P(w^a|x_t) V_{t+1}(S_{t+1}|S_t, x_t, w^a) \right),$$

which can be solved using backward dynamic programming. The expression for the transition probability $P(w^a|x_t)$ can be found in the appendix ("Transition probabilities").

Remark 1 Incorporated in the formulation of the model is the assumption that after a treatment decision x_t at the beginning of time t, patients immediately generate waiting costs in the following queue (if they do not exit the system) after entering that queue in time period $t + 1$. In practice, after a treatment, a patient may require to wait a minimum time lag before a follow-up treatment can be initiated. The model can be extended to cover cases with time lags $d_{i,j}$ (time lag in the transition from queue i to queue j) by allowing u to be negative in $S_{t,j,u}$. For example, $S_{t,j,-2}$ then indicates the number of patients that will enter queue j two time periods from now. Incorporating this time lag changes the system dynamics: patients with $u<0$ cannot be served and we set $C_{t,j,u}\left(S_{t,j,u}, x_{t,j,u}\right) = 0$ for $u<0$, $\forall i \in \mathcal{J}, t \in \mathcal{T}, u \in \mathcal{U}$.

3 Approximate Dynamic Programming

The DP method can be used to solve small instances. These instances particularly do not reflect the complexity and size of a real-life sized instance in a hospital. Computing the exact DP solution is generally difficult and possibly intractable for large problems due to three reasons: (1) the state space $S(t)$ 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 $\mathcal{X}(S_t)$ may be too large to find a good decision for all states within reasonable time, and (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 on the transitions of patients between queues and the external arrivals. To illustrate the size of the state space for our problem, suppose that $\hat{M}$ gives the max number of patients per queue and per number of time periods waiting. The number of states is then given by $\hat{M}^{(|\mathcal{J}|\cdot|\mathcal{U}|)}$.

Various alternatives exist to overcome the intractability problems with DP. The problem size can for example be reduced by aggregating information on resource capacities, patients, and/or time periods. We propose an innovative solution approach within the frameworks of ADP and mathematical programming, which can be used to overcome all three mentioned reasons for intractability of DP for large instances. For more information on ADP, we refer to Powell (2011), who discusses the theory on ADP underlying the approach presented in this section. Our solution approach is based on value iteration with an approximation for the value functions. In this section, we explain this approach in more detail.

First, we discuss the use of a 'post-decision' state as a single approximation for the outcome state. Second, we introduce the method to approximate the value of a state and decision, and third, we explain how we use a 'basis functions' approach in the algorithm to approximate that value. This combination uses an approximation for the expectation of the outcome space, thereby reducing complexity significantly. It also enables calculating the value state by state, making the necessity to calculate the entire state space at once, which was the primary reason of intractability of the exact DP approach, obsolete. Fourth, we explain how we overcome the large decision space for large problem instances with an ILP, and fifth, we reflect on the scalability of our approach.

3.1 Post-decision state

To avoid the problem of a large outcome space and the intractable calculation of the expectation of the 'future' costs, we use the concept of a post-decision state S_t^x (Powell 2011). The post-decision state is the state that is reached, directly after a decision has been made, but before any new information W_t has arrived. It is used as a single representation for all the different states the system can be in the following time period, and it is based on the current pre-decision state S_t and the decision x_t. This simplifies the calculation or approximation of the 'future' costs.

The transitions take place as follows. In addition to the transition function (2), which gives the transition from pre-decision state S_t to pre-decision state S_{t+1}, we introduce a transition function $S^{M,x}(S_t, x_t)$, which gives the transition from the pre-decision state S_t to the post-decision state S_t^x. This function is given by:

$$S_t^x = S^{M,x}(S_t, x_t), \tag{9}$$

with

$$S_{t,j,0}^x = \sum_{i \in \mathcal{J}} \sum_{u \in \mathcal{U}} q_{i,j} x_{t,i,u} \qquad \forall j \in \mathcal{J}, t \in \mathcal{T} \tag{10}$$

$$S_{t,j,U}^x = \sum_{u=U-1}^{U} \left(S_{t,j,u} - x_{t,j,u} \right) \qquad \forall j \in \mathcal{J}, t \in \mathcal{T} \tag{11}$$

$$S_{t,j,u}^x = S_{t,j,u-1} - x_{t,j,u-1} \qquad \forall j \in \mathcal{J}, t \in \mathcal{T}, u \in \mathcal{U} \backslash \{0, U\}. \tag{12}$$

The transition function (9) closely resembles (2), except that the post-decision state is in the same time-period t as the pre-decision state, and the external arrivals to the system are not included in the transition as they are not a result of the decision that is taken. Note that the post-decision state is a direct image of the pre-decision state S_t and the decision x_t.

Due to the patient transfer probabilities, the transition function (9) may result in noninteger 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 pre-decision state and a decision. Hence, the post-decision state is only used as an 'estimate' of the future pre-decision state. The post-decision state will not be used to compute the transition from the pre-decision state in a time period to the pre-decision state in the following time period. Within the ADP algorithm, as presented in the next section, we use the original transition function (2) to compute the pre-decision state in the next time period. As a result, the post-decision state will not cause any pre-decision state to become noninteger.

The actual realizations of new patient arrivals and patient transitions in this time period will be incorporated in the transition to the pre-decision state in the next time period. Note that (9) can be adapted to include pre-defined priority rules like always treating patients with longest waiting times before selecting others within the same queue. This rule is used in our computational experiments as well. For the remainder of this paper, whenever we use the word 'state', we are referring to the pre-decision state.

We rewrite the DP formulation in (8) as

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

where the value function of the post-decision state is given by

$$V_t^x\left(S_t^x\right) = \mathbb{E}\left\{V_{t+1}(S_{t+1})|S_t^x\right\}. \tag{13}$$

To reduce the outcome space for a particular state and decision, we replace the value function for the 'future costs' of the post-decision state $V_t^x\left(S_t^x\right)$ with an approximation based on the post-decision state. We denote this approximation by $\overline{V}_t^n\left(S_t^x\right)$, which we are going to learn iteratively, with n being the iteration counter.

We now have to solve

$$\tilde{x}_t^n = \arg \min_{x_t \in \mathcal{X}_t(S_t)} \left(C_t(S_t, x_t) + \overline{V}_t^{n-1}\left(S_t^x\right)\right), \tag{14}$$

which gives us the decision that minimizes the value $\widehat{v}_t^n$ for state S_t in the n-th iteration. The function $\widehat{v}_t^n$ is given by

$$\widehat{v}_t^n = \min_{x_t \in \mathcal{X}_t(S_t)} \left(C_t(S_t, x_t) + \overline{V}_t^{n-1}\left(S_t^x\right)\right). \tag{15}$$

Note that $V_t^{n-1}\left(S_t^x\right) = 0$ is equivalent to having a standard myopic strategy where the impact of decisions on the future is ignored.

After making the decision $\tilde{x}_t^n$ and finding an approximation for the value in time period t (denoted by $\widehat{v}_t^n$), the value function approximation $\overline{V}_{t-1}^{n-1}\left(S_{t-1}^x\right)$ can be updated. We denote this by

$$\overline{V}_{t-1}^n\left(S_{t-1}^x\right) \longleftarrow U\left(\overline{V}_{t-1}^{n-1}\left(S_{t-1}^x\right), S_{t-1}^x, \widehat{v}_t^n\right). \tag{16}$$

In (16), we update the value function approximation for time period $t-1$ in the n-th iteration with the 'future' cost approximation for time period $t-1$ in the $n-1$-th iteration, the post-state of time period $t-1$, and the value approximation for time period t. The objective is to minimize the difference between the 'future' cost approximation for time period $t-1$ and the approximation $\widehat{v}_t^n$ for time period t with the updating function, as n increases. This is done by using the algorithm presented in the following section.

3.2 The ADP algorithm

We solve (14) recursively. Starting with a set of value function approximations and an initial state vector in each iteration, we sequentially solve a subproblem for each $t \in \mathcal{T}$, using sample realizations of W_t, which makes it a Monte Carlo simulation. In each iteration, we update and improve the approximation of 'future' costs with (16). Consecutively, the subproblems are solved using the updated value function approximations in the next iteration. This is presented in Algorithm 1.

Algorithm 1 The Approximate Dynamic Programming algorithm

```
Step 0.  Initialization
    Step 0a.  Choose an initial approximation $\overline{V}_t^0(S_t)$ for all $t \in \mathcal{T}$ and $S_t$.
    Step 0b.  Set the iteration counter, $n = 1$, and set the maximum number of iterations $N$.
    Step 0c.  Set the initial state to $S_1$.
Step 1.  Do for $t = 1, ..., T$:
    Step 1a.  Solve (14) to get $\tilde{x}_t$.
    Step 1b.  If $t > 1$, then update the approximation $\overline{V}_{t-1}^n\left(S_{t-1}^x\right)$ for the previous post-decision $S_{t-1}^x$ state using
              $\overline{V}_{t-1}^n\left(S_{t-1}^x\right) \longleftarrow U^V\left(\overline{V}_{t-1}^{n-1}\left(S_{t-1}^x\right), S_{t-1}^x, \widehat{v}_t^n\right)$
              where $\widehat{v}_t^n$ is the resulting value of solving (15).
    Step 1c.  Find the post-decision state $S_t^x$ with (9) to (12).
    Step 1d.  Obtain a sample realization $W_{t+1}$ and compute the new pre-decision state with (2).
Step 2.  Increment $n$. If $n \leq N$ go to Step 1.
Step 3.  Return $\overline{V}_t^N(S_t^x)$, $\forall t \in \mathcal{T}$.
```

Using the approximation $\overline{V}_t^N\left(S_t^x\right)$, for all $t \in \mathcal{T}$, we can approximate the value of a post-decision state for each time period. With these approximations, we can find the best decision for each time period and each state, and thus develop a tactical resource capacity and patient admission plan for any given state in any given time period. The difference with the exact DP approach is not only that we now use a value function approximation for the 'future costs', but also that we do not have to calculate the values for the entire state space.

The current set-up of the ADP algorithm is single 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 (Powell 2011), where the algorithm first simulates observations and computes decisions for *all* time periods in one iteration, before updating the value function approximations. This may lead to a faster convergence of the ADP algorithm. We test the use of double pass versus single pass in Sect. 4. More details on double pass can be found in Powell (2011).

3.3 Basis function approach

The main challenge is to design a proper approximation for the 'future' costs $\overline{V}_t^n\left(S_t^x\right)$ that is computationally tractable and provides a good approximation of the actual value to be able to find a suitable solution for the optimization problem of (14). There are various strategies available. A general approximation strategy that works well when the state space and outcome space are large, which generally will be the case in our formulated problem as discussed earlier in this section, is the use of basis functions. We explain the strategy in more detail below.

An underlying assumption in using basis functions is that particular features of a state vector can be identified, that have a significant impact on the value function. Basis functions are then created for each individual feature that reflect the impact of the feature on the value function. For example, we could use the total number of patients waiting in a queue and the waiting time of the longest waiting patient as two features to convert a post-state description to an approximation of the 'future' costs. We introduce

$$\mathcal{F} = \text{set of features,}$$
$$\phi_f(S_t) = \text{basis function for the feature f} \in \mathcal{F} \text{ for the state } \mathrm{S_t}.$$

We now define the value function approximation as

$$\overline{V}_t^n\left(S_t^x\right) = \sum_{f \in \mathcal{F}} \theta_f^n \phi_f\left(S_t^x\right), \qquad \forall t \in \mathcal{T}, \tag{17}$$

where θ_f^n is a weight for each feature $f \in \mathcal{F}$, and $\phi_f\left(S_t^x\right)$ is the value of the particular feature $f \in \mathcal{F}$ given the post-decision state S_t^x. The weight θ_f^n is updated recursively and the iteration counter is indicated with n. Note that (17) is a linear approximation, as it is linear in its parameters. The basis functions themselves can be nonlinear (Powell 2011).

Features are chosen that are independently separable. In other words, each basis function is independent of the other 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. Example features and basis functions are given in Table 3, and we will discuss our selection of basis functions, based on a regression analysis, in Sect. 4.

In each iteration, the value function approximations are updated, as given in (16). In the features and basis functions approach, this occurs through the recursive updating of θ_f^n. Several methods are available to update θ_f^n 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 (Powell 2011). 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.

The method for updating the value function approximations with the recursive least squares method for nonstationary data follows from Powell (2011) and is given in the appendix ("Recursive least squares"). In this method, the parameter α^n determines the weight on prior observations of the value. Setting α^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 α^n to values between 0 and 1 decreases the weight on prior observations (lower α^n means lower weight).

We define the parameter α^n by

$$\alpha^n = \begin{cases} 1, & \text{stationary} \\ 1 - \dfrac{\delta}{n}, & \text{non stationary} \end{cases}, \quad \text{where } n = 1, 2, \ldots, N. \tag{18}$$

where $1 - \frac{\delta}{n}$ is a function to determine α_n that works well in our experiments. We come back to setting α^n (and δ) in Sect. 4.1.

3.4 ILP to find a decision for large instances

In small, toysized problem instances, enumeration of the decision space to find the solution to (14) is possible. For real-life sized problem instances, this may become intractable, as explained in Sect. 2. In this case, we require an alternative strategy to enumeration. In case the basis functions are chosen to be linear with regards to the decision being made (or the resulting post-state description), we can apply ILP to solve (14). The ILP formulation is given in the appendix ("ILP for large instances"), and will be used in Sect. 4.3.

This concludes our theoretical explanation of our solution approach incorporating ADP and ILP. We have formulated an algorithm, an approximation approach involving features to estimate the 'future' costs, a method to update the approximation functions based on new observations, and an ILP formulation to determine the decisions.

In the following section, we will determine the features and various other settings for the ADP algorithm, and discuss the algorithm's performance for small and large instances.

3.5 Scalability

The computational complexity of backward dynamic programming depends on (1) the size of the state space (for each time period we have to evaluate all possible states), (2) the size of the decision space (for each time period and state we have to evaluate all possible decisions), and (3) the size of the outcome space (for each time period, state, and decision, we have to evaluate all possible outcomes). This complexity is reduced drastically with ADP. The size of the state space no longer directly influences the running time of the algorithm. The running time of the ADP algorithm increases linearly with the number of iterations N. Obviously, a larger state space might require more iterations to converge to sufficiently good value function estimates. However, a good value function approximation is able to generalize across states. As we will see in Sect. 4.2.1, we only need a relatively small number of iterations to converge to sufficiently good estimates of the value functions. For a given value function approximation, the required number of iterations N mainly depends on the number of features that we use, which in our problem setting is typically relatively small (say 100, see Sect. 4.1.1). The size of the outcome space also no longer influences the complexity of the ADP algorithm because we avoid computing the expectation by using the post-decision state. The decision space, however, still plays a major role, since we need to evaluate all possible decisions using the ILP from Sect. 3.4. The decision space is bounded by the number of patients waiting and the resource capacities. For larger problem instances, the decision space might be further bounded by setting minimum values for the resource utilization (pre-processing before running the ILP).

4 Computational results

In this section, we test the ADP algorithm developed in Sect. 3. One of the methods prescribed by Powell (2011) is to compare the values found with the ADP algorithm with the values that result from the exact DP solution for small instances. We will use this method in Sects. 4.1 and 4.2. In Sect. 4.3, we study the performance for large instances, where we compare the ADP algorithm with ‘greedy’ planning approaches to illustrate its performance. We first discuss the settings for the ADP algorithm in Sect. 4.1.

4.1 Settings for the ADP algorithm

In this section, we use information from the DP solution to set the basis functions and the parameters for the ADP algorithm. The DP recursions and the ADP algorithm are programmed in Delphi, and for the computational experiments we use a computer with an Intel Core Duo 2.00 GHz processor and 2 GB RAM.

First, we explain the parameters used for the problem instance to calculate the exact DP solution. Second, we explain the selected basis functions and other general settings for the ADP algorithm.

4.1.1 Parameter settings

Some settings in the ADP algorithm, such as the basis functions and double pass or single pass, can be analyzed by comparing the results from the ADP approach with the results from the exact DP approach. The values of the DP can be calculated for extremely small instances only, due to the high dimensions in states and the expectation of the future value in the tactical planning problem. Only for these small instances, we have the opportunity to compare the ADP approximation with the exact DP values. We do not compare the calculated decision policies from both methods, but compare the obtained values. This comparison provides a clear evaluation of the quality of the approximation in the ADP approach for small instances. Since we use exactly the same ADP algorithm for small and real-life sized large instances, this also provides a strong indication of the quality of the approximation accuracy of the ADP approach for large instances (for which we cannot calculate the exact DP value).

For our experiments with small problems, we use the following instance. The routing probabilities $q_{i,j}$ are: $q_{1,2} = 0.8, q_{2,3} = 0.8, q_{1,1} = q_{2,1} = q_{2,2} = q_{3,1} = q_{3,2} = q_{3,3} = 0$. Hence, a patient that is served at Queues 1 or 2 exits the system with probability 0.2, and a patient that is served at Queue 3 will always exit the system. Since there are three queues and there are two periods that a patient can wait: 0 and 1 time period, the state description for a specific time period t becomes: $\left[S_{t,1,0}, S_{t,1,1}, S_{t,2,0}, S_{t,2,1}, S_{t,3,0}, S_{t,3,1}\right]$. The exact DP-problem is restricted by limiting the number of patients that can be waiting in each queue to 7. The state holding the most patients is thus $[7, 7, 7, 7, 7, 7]$. If there are transitions or new arrivals that result in a number greater than 7 for a particular queue and waiting time, the number for that particular entry is set to 7. So if, after transitions, we obtain $[3, 1, 6, 8, 5, 4]$, this state is truncated to $[3, 1, 6, 7, 5, 4]$. In the same way, the states in the ADP are also restricted for comparison, even though this is not necessary. For large instances, when comparison with an exact DP solution is impossible, this state truncation method is not used. The state truncation may affect the ADP-approximation slightly, as it introduces nonlinearity around the edges of the state space. Using the number of time periods, the truncated state space, the number of queues, and the maximum number of time periods waiting, there are $8 \cdot 8^{(3 \cdot 2)} = 2,097,152$ entries to be calculated. The weights θ^n in the value function approximations are initialized to $\theta^0 = 1$ for all time periods, and the matrix $B^0 = \epsilon I$ as explained in Sect. 7.3. All other parameters are given in Table 1.

In the appendix ("ADP settings"), we provide computational results for selecting a proper basis function, to decide between using a double pass or single pass version of the ADP algorithm, and to select the right step size α^n. For the remainder of the computational experiments, unless mentioned otherwise, we use a double pass algorithm and determine α with the nonstationary version of (18) with $\delta = 0.99$. For the basis functions, we choose to use the features 'The number of patients in queue j that are u periods waiting'. 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, see the appendix ("ADP settings"), and they can be straightforwardly obtained from the state or post-state description. In case there is no independent

Table 1 The parameters that characterize the test instance

Parameter	Description	Used values
T	The number of time periods	$8, \mathcal{T} = \{1, 2, \ldots, 7, 8\}$
R	The number of resource types	1
J	The number of queues	$3, \mathcal{J} = \{1, 2, 3\}$
U	The number of periods waiting	$2, \mathcal{U} = \{0, 1\}$
$s_{j,r}$	Expected service time from resource type $r \in \mathcal{R}$ for a patient in queue $j \in \mathcal{J}$	1
$\eta r, t$	Resource capacity for resource type $r \in \mathcal{R}$ in time $t \in \mathcal{T}$ in same unit as $s_{j,r}$	6
$\lambda_{1,t}$	Poisson parameter for new demand in the Queue 1 in time period $t \in \mathcal{T}$	5
$c_{t,j,u}$	Costs per patient waiting in a queue $j \in \mathcal{J}$, for $u \in \mathcal{U}$ time periods, in time period $t \in \mathcal{T}$	$\frac{(u+1)}{j}$

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 θ_f^n may vary, but the feature value $\phi_f\left(S_t^x\right)$ of this constant is always 1, independent of the state S_t^x.

4.2 Comparison of ADP, DP and greedy approaches on small instances

In this section, the values calculated with the ADP approach are compared with the exact DP solution and two greedy approaches.

4.2.1 Convergence of the ADP algorithm

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 standard deviation 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 in Sect. 4.1.1.

For two initial states, Fig. 2 illustrates that the calculated values with the ADP-algorithm (with $\delta = 0.99$ and double pass) converge to the values calculated with the exact DP approach as the number of iterations grow. In the first iterations, the ADP-values may be relatively volatile, due to the low value for α and thus the high impact of a new observation on the approximation. When the number of iterations

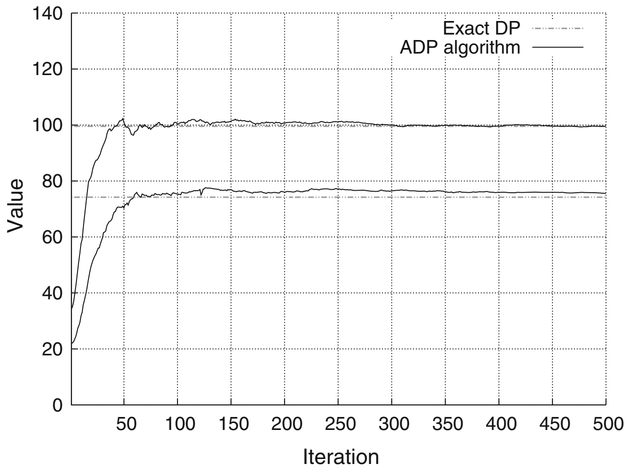

Fig. 2 Example for two initial states. The values approximated with the ADP algorithm (with $\delta = 0.99$ and double pass) converge to the values from the exact DP approach

increases, the weight on prior observations increases as α increases in (18), and the ADP approximations become less volatile.

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 0.439 s, which is 0.0001 % of the calculation time for the exact DP solution. Obviously the calculation times depend on the used computational power, but these results indicate that solving a toy problem with the exact DP approach is already very time intensive, and solving such a problem with the ADP approximative approach is significantly faster.

4.2.2 Comparing the use of the value function approximation, with DP and two greedy approaches

In the section above, we have evaluated the performance of the ADP algorithm to find the feature weights used in the value function approximation (17). After the ADP algorithm has established the feature weights θ^n that accurately approximate the value associated with a state and a decision, these weights for all time periods are fixed and used to calculate planning decisions for each time period. In this section, we evaluate the accuracy of the ADP approach by comparing the values obtained with the ADP approach, the DP approach, and two greedy approaches.

The two greedy approaches are rules that can be used to calculate a planning decision for a particular state and time period. We call the two approaches ‘HighestNumberOfWaitingPatientsFirst’ and ‘HighestCostsFirst’. In the greedy approach ‘HighestNumberOfWaitingPatientsFirst’, the queue with the highest number of waiting patients is served until an other queue has the highest number of waiting patients, or until resource capacity constraints do not allow serving another patient of this queue anymore. After that, the next highest queue is served in the same way, until all queues are served and/or resource capacity constraints do not allow serving another patient anymore. In the greedy approach ‘HighestCostsFirst’, the queue with the highest costs (calculated with the cost function and the state description) is served until an other queue has the highest costs, or until resource capacity constraints do not allow serving another patient of this queue anymore. After that, the next highest queue is served in the same way, until all queues are served and/or resource capacity constraints do not allow serving another patient anymore.

To compare the value calculated with the four approaches, we calculate a planning decision for each separate time period as follows. As a first step, we generate an initial state for the first time period in time horizon $\mathcal{T}$. We can find the exact DP value associated with this initial state from the already calculated DP solution. To establish the values for the ADP approach and the two greedy approaches, we use simulation as follows. We use the value function approximations from the ADP approach and the described methods from the greedy approaches, to establish a planning decision for the chosen initial state in the first time period. Then simulate the outcomes for patient transfers and patient arrivals. This leads to a particular state in the following time period for which we can establish the planning decision using the ADP approach and greedy approaches. These steps are repeated until the end of the time horizon $\mathcal{T}$. We sum the values associated with each state in each time period in the time horizon $\mathcal{T}$, to obtain the value for the initial state in the first time period. These values are then compared between the different approaches. By following this method, we can properly evaluate and compare the ADP approach in a wide range of possible outcomes for patient transfers and patient arrivals. When one aims to establish a tactical plan for a complete time horizon upfront, the random patient transfers and patient arrivals are replaced by the expectation for these processes.

We randomly choose a set of 5000 initial states, that we each simulate with 5000 sample paths for the ADP approach and the two greedy approaches. We calculate the relative difference with the DP value for each of the 5000 initial states. Figure 3 displays the average over all initial states. The graph illustrates that the ADP approach provides a relatively accurate approximation for the value of a particular state, and the approximation is significantly better than two greedy approaches. The value resulting from the policy (the value function approximations) obtained with the ADP approach is very close to the values obtained with the optimal policy (found with the exact DP approach). Consequently, the fast and accurate ADP approach is very suitable to determine tactical planning decisions for each time period, and thus to establish a tactical plan for a complete time horizon.

These results indicate that the ADP algorithm is suitable for the tactical resource capacity and patient admission planning problem.

4.3 Performance of the ADP algorithm on large instances

In the previous sections, we analyzed the performance of the ADP algorithm for small, toysized problems to compare the results with the DP approach. In this

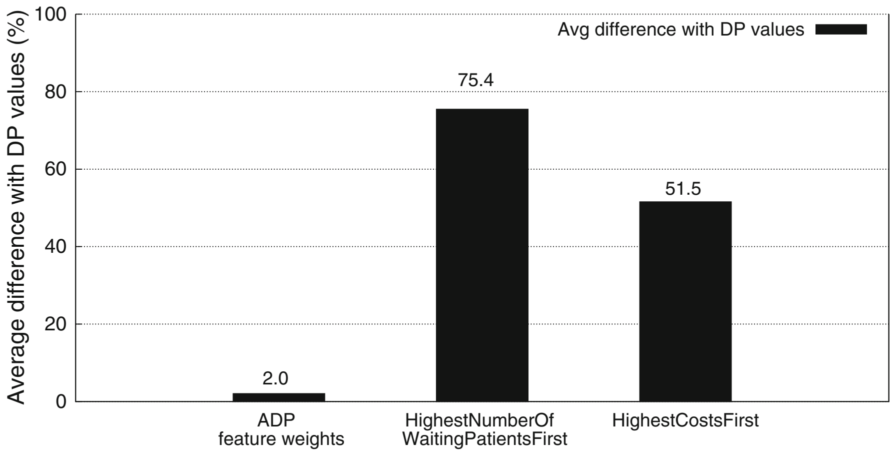

Fig. 3 The average difference with the DP value when using the feature weights from ADP or the two greedy approaches to develop a tactical plan. The average value calculated with the ADP approach is 92.5

section, we investigate the performance of the ADP algorithm for large, real-life sized instances that we encountered in practice. The instances we generate in this section are based on the typical size of tactical planning problems we have encountered in several Dutch general hospitals. We have interacted with those hospitals in understanding and defining the tactical planning problem as part of the overarching research project LogiDOC, at the Center for Healthcare Operations Improvement and Research (CHOIR) at the University of Twente. Typically, we observe that the tactical plan is planned week by week, suggesting that weeks should be used as the time period. Capacity and resource usage are often state by minutes within that week. Since for large instances, computation of the exact DP approach is intractable, we evaluate the performance of the ADP algorithm with the two greedy approaches as introduced in Sect. 4.2.2.

4.3.1 Parameters for large problem instances

As explained in Sect. 2, for large instances, the decision space becomes too large to allow for complete enumeration. Hence, we use an ILP to compute the optimal decision and use a CPLEX 12.2 callable library for Delphi to solve the ILP, and tolerate solutions with an integrality gap of 0.01 %.

The parameters to generate the large instances are given in Table 2. When multiple entries are listed, we randomly choose one for each variable. For example, for each initial queue in a care process, we randomly pick the Poisson parameter for new demand from the set 1, 3, or 5. The resource capacities $\eta_{r,t}$ for each resource $r \in \mathcal{R}$ and $t \in \mathcal{T}$ are selected from the given set, this means that we can for example have: $\eta_{1,1} = 1200$, $\eta_{1,2} = 0$, $\eta_{1,3} = 1200$, $\eta_{2,1} = 3600$, $\eta_{2,2} = 3600$, and $\eta_{2,3} = 1200$ can be 0. As real-life instances may have changing patient arrivals and changing resource capacities over time, we vary these parameters over the time

Table 2 The parameters that characterize the large test instances

Parameter	Description	Used values
T	The number of time periods	$8, \mathcal{T} = \{1, 2, \ldots, 8\}$
R	The number of resource types	4
J	The number of queues	$40, \mathcal{J} = \{1, 2, \ldots, 40\}$
U	The number of periods waiting	$4, \mathcal{U} = \{0, 1, 2, 3\}$
$s_{j,r}$	Expected service time from resource type $r \in \mathcal{R}$ for a patient in queue $j \in \mathcal{J}$ (four value sets)	$\{10, 15, 20\}, \{30, 45, 60\}$, $100, 120, 140\}, \{200, 220, 240\}$
$\eta_{r,t}$	Resource capacity for resource type $r \in \mathcal{R}$ in time $t \in \mathcal{T}$ in same unit as $s_{j,r}$	$\{0, 750, 1000, 1200, 1250, 2000, 3600, 5000, 8750, 9600, 10000, 17600\}$
$q_{i,j}$	The routing probabilities between queue $i,j \in \mathcal{J}$	$\{0, 0.25, 0.5, 0.75, 1\}$
$\lambda_{1,t}$	Poisson parameter for new demand in the first queue of each care process in time period $t \in \mathcal{T}$	$\{1, 3, 5\}$
$c_{t,j,u}$	Costs per patient waiting in a queue $j \in \mathcal{J}$, for $u \in \mathcal{U}$ time periods, in time period $t \in \mathcal{T}$	$\frac{(u+1)}{j}$

periods for each queue and each resource respectively. In contrast with the exact DP approach, truncation of the state space is not required for the ADP algorithm, and we will not truncate the state space in the experiments for large instances. We truncate the initial starting state, to ensure that it is in line with the selected resource capacities and resource requirements. To generate the initial states, we randomly pick the number of patients, for each queue and each number of time periods waiting, from the set $[0, 1, \ldots, 4]$. This set is bounded to align the initial state with the generated instance for the available resource capacity with the settings in the Table 2.

The weights θ^n in the value functions are initialized to $\theta^0 = 1$ for all time periods, and the matrix $B^0 = \epsilon I$ as explained in the appendix ("Recursive least squares").

4.3.2 Comparison with greedy approaches

After running the ADP algorithm for $N = 100$ iterations, we fix the established feature weights θ^n, and use these to calculate tactical planning decisions in each time period. In this section, we compare the use of the feature weights calculated in the ADP algorithm with the two greedy approaches introduced in Sect. 4.2.2: 'HighestNumberOfWaitingPatientsFirst' and 'HighestCostsFirst'. For the comparison, we use the same simulation approach as explained in Sect. 4.2.2, but for larger instances, we simulate 30 initial states over 10 different generated instances. Similar to Sect. 4.2.2, we perform 5000 simulation runs per initial state. The relative difference between 'HighestNumberOfWaitingPatientsFirst' and the ADP approach is 50.7 %, and the relative difference between 'HighestCostsFirst' and the ADP approach is 29.1 %. The average value calculated with the ADP approach is 129.0. The lower average value from the ADP approach indicates that the ADP approach develops tactical plans resulting in lower costs than the two greedy approaches for large instances. The lower values indicate that the ADP approach supports and improves tactical planning decision making, and therefore we can conclude that the ADP approach is a suitable method to calculate a tactical plan for real-life sized instances.

Running the ADP algorithm for a given initial state (with $N = 100$) takes approximately 1 h and 5 min for the large instance. This seems reasonable for finding the feature weights that approximate the value functions for 40 queues and eight time periods. The feature weights that are calculated for the complete time horizon can be used to adjust the tactical plan in later time periods, as time progresses. Hence, the algorithm does not have to be run on a daily or even weekly basis. Since the algorithm converges fast, one may further decrease the number of iterations, resulting in lower runtimes.

4.3.3 Benefit of considering future costs through the ADP approach

Compared to the greedy approach 'HighestCostsFirst', the ADP approach offers an advantage by also considering costs of the future effects of the evaluated decision.

The benefits of this advantage seem especially strong, when parameters such as resource capacity and patient arrivals change over time periods. The finite time horizon in the ADP approach allows for setting time dependent parameters for the problem instance, thereby ensuring that changing parameters over time are incorporated in the decision making. To illustrate the additional benefits of considering future costs in instances where parameters change over time, we have conducted the following experiment.

We generated instances with Table 2, but we limited the number of resource types required for each queue to 1 resource type only. We set the total resource capacities for each type as follows: $\sum_{t=1}^{T} \eta_{1,t} = 6000$, $\sum_{t=1}^{T} \eta_{2,t} = 10,000$, $\sum_{t=1}^{T} \eta_{3,t} = 30,000$, and $\sum_{t=1}^{T} \eta_{4,t} = 70,000$. Next, we set a number of entries for the resource capacities $\eta_{r,t}$ for $r \in \mathcal{R}$ and $t \in \mathcal{T}$ to zero. The total resource capacity for each resource type for the full time horizon is evenly distributed over the remaining queues that are not set to zero.

We compare three scenarios: setting 2, 8 and 14 entries of the total 32 entries for $\eta_{r,t}$ to zero in the complete time horizon $t = 1, \ldots, T$. For the comparison of the three scenarios, we use the same simulation approach as explained in Sect. 4.2.2, but we simulate eight initial states. We perform 5000 simulation runs per initial state. In each of the three scenarios, we use the same instances and the same eight initial states for our calculations and simulation.

The results in Fig. 4 illustrate that a higher variation of resource capacities in the time horizon, gives a higher benefit of using the ADP approach compared to the 'HighestCostsFirst' approach. These results indicate that the benefit of considering future costs in making a tactical planning decision increases when volatility in resource capacities increases.

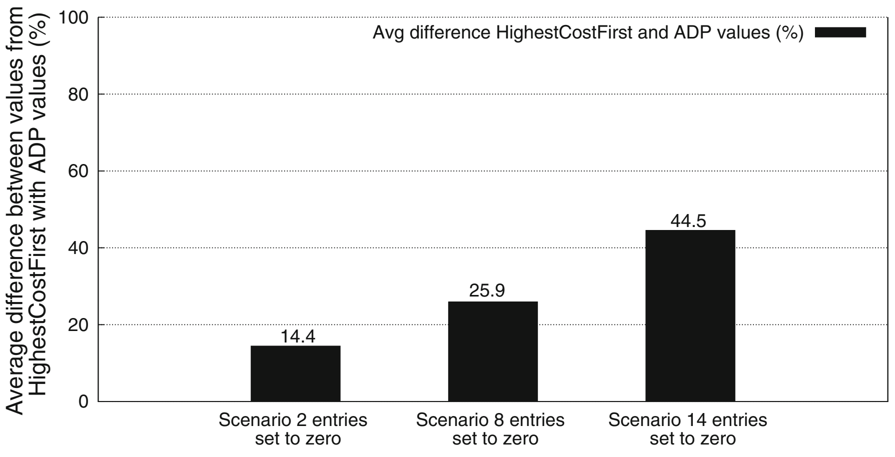

Fig. 4 The average difference between the value calculated by using the feature weights from ADP and the value calculated by using the greedy approach 'HighestCostsFirst'. The average value calculated with the ADP approach is 165.1

5 Implementation

The ADP approach can be used to establish long-term tactical plans (e.g., three month periods) for real-life instances in two steps. First, N iterations of the ADP algorithm have to be performed to establish the feature weights for each time period $t \in \mathcal{T}$. Implicitly, by determining these feature weights, we obtain and store the value functions as given by (15) and (17) for each time period. Second, these value functions can be used to determine the tactical planning decision for each state and time period by enumeration of the decision space or the ILP as introduced in Sect. 3.4.

For each consecutive time period in the time horizon, the state transitions are calculated by using the state in the current time period, the decision calculated with the value function approximations, the expected number of patient arrivals and patient transfers between the queues. Subsequently, the value function approximations are used to determine the tactical planning decision for the new state in the following time period. This is repeated for all successive time periods until the end of the time horizon.

The actual tactical plan is implemented using a rolling horizon approach, in which for example tactical plans are developed for three consecutive months, but only the first month is actually implemented and new tactical plans are developed after this month. The rolling horizon approach is recommended for two reasons. First, the finite horizon approach, apart from the benefits it provides to model time dependent resource capacities and patient demand, may cause unwanted and short-term focused behavior in the last time periods. Second, recalculation of tactical plans after several time periods have passed, ensures that the most recent information on actual waiting lists, patient arrivals, and resource capacities is used.

6 Conclusions

We provide a stochastic model for tactical resource capacity and patient admission planning problem in healthcare. Our model incorporates stochasticity in two key processes in the tactical planning problem, namely the arrival of patients and the sequential path of patients after being served. A Dynamic Programming (DP) approach, which can only be used for extremely small instances, is presented to calculate the exact solution for the tactical planning problem. We illustrate that the DP approach is intractable for large, real-life sized problem instances. To solve the tactical planning problem for large, real-life sized instances, we developed an approach within the frameworks of Approximate Dynamic Programming (ADP) and Mathematical Programming.

The ADP approach provides robust results for small, toyproblem instances and large, real-life sized instances. When compared with the exact DP approach on small instances, the ADP algorithm provides accurate approximations and is significantly faster. For large, real-life sized instances, we compare the ADP algorithm with the values obtained with two greedy approaches, as the exact DP approach is intractable for these instances. The results indicate that the ADP algorithm performs better than the two greedy approaches, and does so in reasonable run times. The real-life sized, large instances used to test the ADP approach in this paper are developed based on examples encountered in several Dutch hospitals. As a result, the instances used are of a comparable network structure and size as tactical planning problems encountered in practice. We used the generated instances to illustrate the applicability of our ADP approach to support tactical planning problems in practice. As a next step, we aim to apply this ADP approach to actual real-life hospital data.

We conclude that ADP is a suitable technique for developing tactical resource capacity and patient admission plans in healthcare. The developed model incorporates the stochastic processes for (emergency) patient arrivals and patient transitions between queues in developing tactical plans. It allows for time dependent parameters to be set for patient arrivals and resource capacity in order to cope with anticipated fluctuations in demand and resource capacity. The ADP model can also be used as for readjusting existing tactical plans, after more detailed information on patient arrivals and resource capacities are available (for example when the number of patient arrivals were much lower than anticipated in the last week). The developed ADP model is generic, where the objective function can be adapted to include particular targets, such as targets for access times, monthly ‘production’ or resource utilization. Also, the method can be extended with additional constraints and stochastic elements can be added to suit the hospital situation at hand. It can potentially be used in industries outside healthcare. Future research may involve these extensions, and may also focus on further improving the approximation approach, developing tactical planning methods to adjust a tactical plan when it is being performed, or using the ADP approach for other tactical planning objectives.

Acknowledgments This research has been supported by the Dutch Technology Foundation STW, applied science division of NWO and the Technology Program of the Ministry of Economic Affairs.

Appendix

Transition probabilities

To describe the transition probabilities $P(w^a|x_t)$ used in the model in Sect. 2, we introduce the vector w^d, consisting of elements w_j^d representing the number of patients leaving queue j, for all j in $\mathcal{J}$, and use w_0^d for the number of patients arriving from outside the system (leaving the ‘outside’). To administer all possible transitions, we introduce the elements w_{ij} representing a realization for the number of patients that are transferred from queue i to queue j after service at queue i. w_{0j} represents the realization of the number of external arrivals at queue j. w_{j0} represents the number of patients leaving the hospital after treatment at queue j. Remember from Sect. 2 that the vector w^a represents a realization of the number of patients arriving at each queue.

Under Assumption 1, the transition process follows a multinomial distribution with the parameters $q_{j,i}$ with $i,j \in \mathcal{J}$, and $q_{j,0} = 1 - \sum_{i \in \mathcal{J}} q_{j,i}$ for patients leaving the hospital. From Boucherie and Van Dijk (1991), we obtain

$$P(w^a | x_t) = \sum_{w_0^d = 0}^{\infty} P(w_0^d) \times$$

$$\sum_{\left\{\begin{array}{c} w_{ij}, i = 0, \ldots, J, j = 0, \ldots, J: \\ w_j^d = \sum_{u \in \mathcal{U}} x_{t,j,u}, j = 1, \ldots, J, \\ \sum_{j=0}^{J} w_{ij} = w_i^d, i = 0, \ldots, J, \\ \sum_{i=0}^{J} w_{ij} = w_j^a, j = 0, \ldots, J, \\ w_{ij} \geq 0, w_{ij} = 0 \text{ if } p_{i,j} = 0, w_{00} = 0. \end{array}\right\}} \prod_{i=0}^{J} \binom{w_i^d}{w_{i0}, \ldots, w_{iJ}} \prod_{j=0}^{J} p_{i,j}^{w_{ij}},$$

where $p_{i,j}$ represents the transfer probability for a single patient from queue $i \in \mathcal{J}$ to queue $j \in \mathcal{J}$. The second summation in this equation sums over all possible realizations of w_{ij} that result in the vector w^a, given decision x_t and the realization for the number of patients w_0^d arriving to the hospital system from outside (given by the first summation). Realizations of w_{ij} are subject to six constraints. First, the summation in the second line computes the number w_j^d of patients leaving each queue $j \in \mathcal{J}$. This is equal to the total number of patients we decided to treat in queue j, which is given by $\sum_{u \in \mathcal{U}} x_{t,j,u}$. Second, the total number of patients leaving queue $i \in \mathcal{J}$ should be equal to w_i^d. The third constraint ensures that the number of patients that arrive to a queue in the realization is equal to the number of arrivals in the vector w^a for which the transition probability is calculated. The last three constraints are in line with the dynamics of our model description. The complete derivation can be found in Boucherie and Van Dijk (1991), Section 3.3.2 on independent routing in open queueing networks without blocking.

With Assumption 2, we obtain

$$P(w_0^d) = \frac{\lambda_{0,t}^{w_0^d}}{w_0^d!} e^{-\lambda_{0,t}}, \text{with } \lambda_{0,\mathrm{t}} = \sum_{\mathrm{j}=1}^{\mathrm{J}} \lambda_{\mathrm{j,t}},$$

and

$$p_{i,j} = \begin{cases} q_{i,j}, & \text{when i} = 1, \ldots, \text{J}, \text{j} = 0, \ldots, \text{J}, \\ \dfrac{\lambda_{j,t}}{\lambda_{0,t}}, & \text{when i} = 0, \text{j} = 1, \ldots, \text{J}, \\ 0, & \text{when i} = 0, \text{j} = 0. \end{cases}$$

ILP for large instances

$$\min_{x_t \in \mathcal{X}_t(S_t)} \left(C_t(S_t, x_t) + \sum_{f \in \mathcal{F}} \theta_f \phi_f\left(S_t^x\right) \right),$$

subject to

$$S_{t,j,0}^x = \sum_{i \in \mathcal{J}} \sum_{u \in \mathcal{U}} q_{i,j} x_{t,i,u} \quad \forall j \in \mathcal{J}, t \in \mathcal{T}, \tag{19}$$

$$S_{t,j,U}^x = \sum_{u=U-1}^{U} \left(S_{t,j,u} - x_{t,j,u}\right) \quad \forall j \in \mathcal{J}, t \in \mathcal{T}, \tag{20}$$

$$S_{t,j,u}^x = S_{t,j,u-1} - x_{t,j,u-1} \quad \forall j \in \mathcal{J}, t \in \mathcal{T}, u \in \mathcal{U} \backslash \{0, U\}, \tag{21}$$

$$x_{t,j,u} \leq S_{t,j,u} \quad \forall j \in \mathcal{J}, t \in \mathcal{T}, u \in \mathcal{U}, \tag{22}$$

$$\sum_{j \in \mathcal{J}^r} s_{j,r} \sum_{u \in \mathcal{U}} x_{t,j,u} \leq \eta_{r,t} \quad \forall r \in \mathcal{R}, t \in \mathcal{T}, \tag{23}$$

$$x_{t,j,u} \in \mathbb{Z}_+ \quad \forall j \in \mathcal{J}, t \in \mathcal{T}, u \in \mathcal{U} \tag{24}$$

Constraints (19) to (21) ensure that the waiting list variables are consistently calculated. Constraint (22) ensures that not more patients are served than the number of patients on the waiting list. Constraint (23) is introduced such that not more resource capacity of each resource type $r \in \mathcal{R}$ is used than is available, and Constraint (24) is an integrality constraint.

Recursive least squares

The method for updating the value function approximations with the recursive least squares method for nonstationary data is explained in detail in Powell (2011). The equations used in our solution approach are given below.

The weights θ_f^n, for all $f \in \mathcal{F}$, are updated each iteration (n is the iteration counter) by

$$\theta_f^n = \theta_f^{n-1} - H_n \phi_f\left(S_t^x\right)\left(\overline{V}_{t-1}^{n-1}\left(S_{t-1}^x\right) - \hat{v}_t^n\right), \qquad \forall f \in \mathcal{F},$$

where H^n is a matrix computed using

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

B^{n-1} is an $|\mathcal{F}|$ by $|\mathcal{F}|$ matrix, which is updated recursively using

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

The expression for γ^n is given by

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

B^n is initialized by using $B^0 = \epsilon I$, where I is the identity matrix and ϵ is a small constant. This initialization especially works well when the number of observations is large (Powell 2011). The parameter α^n determines the weight on prior observations of the value, and it is discussed in Sects. 3 and 4.1.

ADP settings

This appendix describes the experiments done to set the ADP algorithm. For the experiments in this section, we have used the instance as described in Sect. 4.1. This appendix describes the selection of the basis functions, double pass and setting α.

Selection of basis functions

In Sect. 3, we introduced basis functions to approximate the future value of a particular decision in a particular state. Basis functions are used because of their relative simplicity. The selection of the features however, requires careful design. The challenge in this careful design is to make sure the choice of basis functions actually contributes to the quality of the solution. The basis functions can be observed as independent variables in the regression literature (Powell 2011). Hence, to select a proper set of basis functions that have significant impact on the value function, we use a regression analysis. In the regression analysis, the dependent variables are the computed values in the exact DP approach for the first time period, and the independent variables are the basis functions calculated from the state description.

Table 3 shows the regression results on various basis functions. The R^2 depicts the variation in the value that is explained by a regression model that uses the features as mentioned in the table as independent variables. The higher R^2, the better suitable the basis functions are for predicting (and thus approximating) the value. One can observe that the features with high level of detail about the state description score significantly better (are higher in the ordered table). Obviously, in addition to the basis functions in Table 3, a significant number of alternatives are available.

For our ADP-model, we choose to use the features ‘The number of patients in queue j that are u periods waiting’ from the list in Table 3. These basis function explain a large part of the variance in the computed values with the exact DP approach ($R^2 = 0.954$), and the basis functions can be straightforwardly obtained from the state or post-state description. We choose these functions as they seize the highest level of detail on the state description, and therefore are likely to provide high quality approximations.

Double pass

In Sect. 3 we introduced the possible use of double pass, where the algorithm first steps through all time periods before updating the value functions. Our experiments

Table 3 The basis functions and their R^2 regression on the given value function. In each regression, a constant is added as a variable. All R^2 values are obtained with significance of 0.000, indicating a good fit of the model. The third column '# vars' indicates the number of variables when the particular basis function is used

Features	Basis functions	# vars	R^2
The number of patients in queue j that are u periods waiting	$S_{t,j,u}, \forall j \in \mathcal{J}, \forall u \in \mathcal{U}, t = 1$	$J \times U$	0.954
Combination of the total number of patients in queue j and the sum of the number of time periods all patients are waiting in queue j	$\sum_{u=0}^{U} S_{t,j,u}$ and $\sum_{u=0}^{U} u \cdot S_{t,j,u}, \quad \forall j \in \mathcal{J}, t = 1$	$2 \times J$	0.954
Combination of the total number of patients in queue j and the longest waiting time currently in queue j	$\sum_{u=0}^{U} S_{t,j,u}$ and $\max\limits_{u \in \mathcal{U}} S_{t,j,u}, \quad \forall j \in \mathcal{J}, t = 1$	$2 \times J$	0.954
The total number of patients in queue j	$\sum_{u=0}^{U} S_{t,j,u}, \forall j \in \mathcal{J}, t = 1$	J	0.950
The sum of the number of time periods all patients are waiting in queue j	$\sum_{u=0}^{U} u \cdot S_{t,j,u}, \forall j \in \mathcal{J}, t = 1$	J	0.879
The longest waiting time currently in queue j	$\max\limits_{u \in \mathcal{U}} S_{t,j,u}, \forall j \in \mathcal{J}, t = 1$	J	0.199
The average waiting time in queue j	$\frac{\sum_{u=0}^{U} u \cdot S_{t,j,u}}{\sum_{u=0}^{U} S_{t,j,u}}, \forall j \in \mathcal{J}, t = 1$	J	0.033

confirm that double pass leads to faster convergence of the ADP algorithm than single pass.

To illustrate the effect, we compare the values from the exact DP solution with the found ADP values for 5000 randomly generated states. The ADP algorithm uses the recursive least squares method for nonstationary data, with $\delta = 0.95$ in (18). To evaluate the speed of the ADP algorithm, we display the number of iterations required until the algorithm is within 5 % of the DP value, so either 95 % or 105 % of the DP value for a particular state. The average number of iterations before the ADP value is within 5 % of the DP value for the 5000 states is 1131.0 when double pass is not used, and 100.3 when double pass is used. Hence, double pass is a significantly faster method to get an accurate approximated value. This effect can also be observed in Fig. 5 for a single state. Also for other values of δ in (18), we find that the use double pass leads to faster convergence to the DP value. For the remainder of our experiments, we use double pass.

Setting α

The parameter α is set in (18). When $\alpha = 1$ is chosen, the recursive least square method for stationary data is selected, and equal weight is given to each observation. Because the ADP algorithm is initialized with given arbitrary weights for θ^n and B^n, there is a 'warm-up period' before the weights are properly iterated and getting closer to the actual value. Hence, it seems useful to put less emphasis on the first observations, and more emphasis on later ones. To achieve this, the recursive least squares method for nonstationary data is used, as explained in Sect. 3.3.

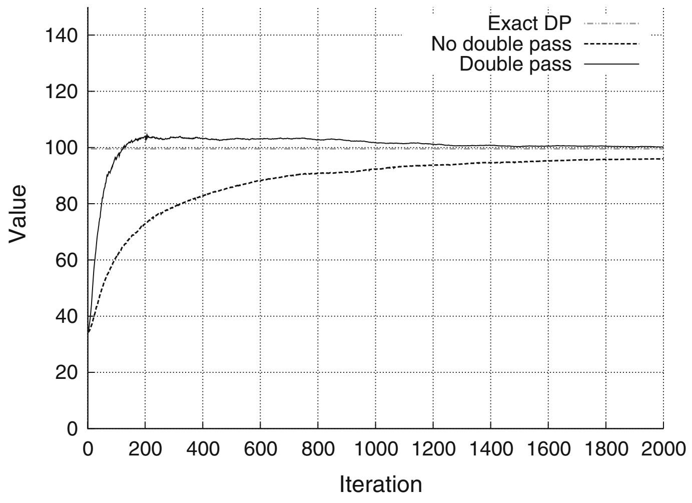

Fig. 5 The values approximated with the ADP algorithm (Settings: recursive least squares for nonstationary data and $\delta = 0.95$) and calculated with the exact DP approach for initial state [2,7,5,1,7,4]. These *graphs* illustrate the significantly faster convergence when double pass is used

To find a good value for δ, we compare the values from the exact DP solution with the found ADP values for 5000 randomly generated states. We compare the number of iterations required until the algorithm is within 5 % of the DP value, and the average difference between the ADP value and the DP value [(ADP value−DP value)/DP value] for various settings of δ. Figure 6 shows the results of these experiments. Note that δ cannot be equal to 1, because this would result in $\alpha^1 = 0$ and a division by 0 in (7.3) in the first iteration.

The recursive least squares method for *stationary data* requires 83 runs to reach a value within 5 % of the DP value, and has an average difference of 2.2 % (standard deviation of 2.7 %) after 2500 iterations. The recursive least squares method for *nonstationary data* achieves similar average difference, but in fewer iterations. We explain the results for the recursive least square method for nonstationary data below.

The left side of Fig. 6 shows that when δ is closer to 1 (and α^1 is close to 0), the number of iterations required to reach a value within 5 % of the DP value is significantly lower. This is due to the structure of (18), where a higher δ causes a lower α, which puts less emphasis on prior observations. In the first iteration, the prior observations are initializations, done by the modeler and independent of the instance or state. Hence, a 'warm-up period' is required to 'forget' the initializations and approximate the actual values. From the experiments it is clear that setting $\delta \geq 0.6$ decreases the warm-up period significantly, and results in stable performance on the average and standard deviation of the difference. Setting $\delta \leq 0.5$ results in unstability in the matrix operations of the nonstationary least squares method, resulting in strongly decreasing average difference (resulting in longer runtimes required to get to proper results). We obtain even more stringent conclusions if we observe the results for average and standard deviation of the difference: 200

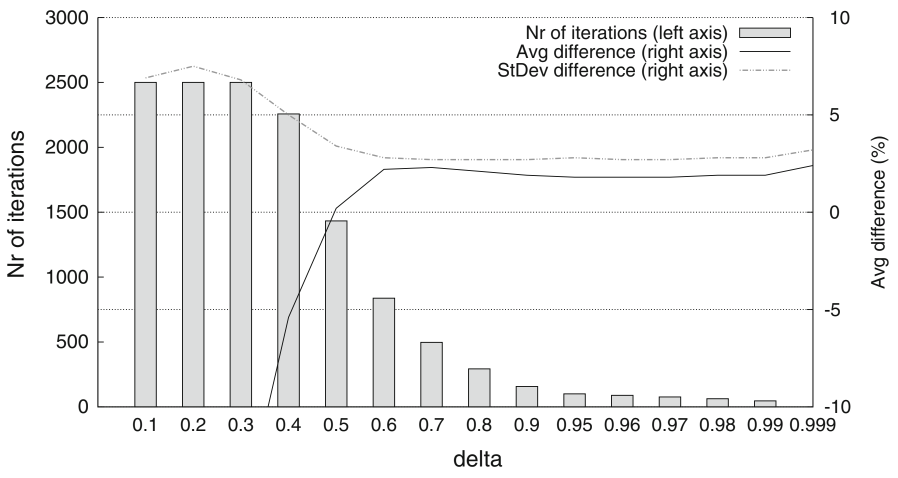

Fig. 6 The average and standard deviation of the difference between the ADP value and the DP value (ADP value−DP value divided by the DP value) are depicted with lines (on the *right axis*) and the average number of runs required until the algorithm is within 5 % of the DP value is given with block diagrams (on *left axis*). The number of iterations are bounded to 2500 for this experiment

iterations after the ADP algorithm finds values within 5 % of the DP value. It appears that $\delta \geq 0.8$ gives the best results. Note that with the division of δ by the iteration number n in (18), α increases fast. After 10 iterations, $\alpha = 0.901$, with $\delta = 0.99$.

From the above it is clear that setting $\delta = 0.99$ results in stable, relatively good performance. For the remainder of our experiments, we use this setting which approximates the DP value within 5 % in an average of 46.1 iterations and accurately with an average difference with the DP value of 1.9 % and standard deviation of this difference of 2.8 % after 2500 iterations for 5000 states.

References

Bellman RE (1957) Dynamic programming. Princeton University Press, Princeton NJ

Boucherie RJ, van Dijk NM (1991) Product forms for queueing networks with state-dependent multiple job transitions. Adv Appl Probab 23(1):152–187

Cardoen B, Demeulemeester E (2008) Capacity of clinical pathways—a strategic multi-level evaluation tool. J Med Syst 32(6):443–452

Erdelyi A, Topaloglu H (2010) Approximate dynamic programming for dynamic capacity allocation with multiple priority levels. IIE Trans 43(2):129–142

Every NR, Hochman J, Becker R, Kopecky S, Cannon CP (2000) Critical pathways: a review. Circulation 101(4):461–465

Garg L, McClean S, Meenan B, Millard P (2010) A non-homogeneous discrete time Markov model for admission scheduling and resource planning in a cost or capacity constrained healthcare system. Health Care Manag Sci 13:155–169

Graves SC (1986) A tactical planning model for a job shop. Oper Res 34(4):522–533

Hall RW (2006) Patient flow: reducing delay in healthcare delivery. Springer, Berlin

Hulshof PJH, Boucherie RJ, Hans EW, Hurink JL (2013) Tactical resource allocation and elective patient admission planning in care processes. Health Care Manag Sci 16(2):152–166

Hulshof PJH, Kortbeek N, Boucherie RJ, Hans EW, Bakker PJM (2012) Taxonomic classification of planning decisions in health care: a structured review of the state of the art in OR/MS. Health Syst 1(2):129–175

Kapadia AS, Vineberg SE, Rossi CD (1985) Predicting course of treatment in a rehabilitation hospital: a Markovian model. Comput Oper Res 12(5):459–469

Maxwell MS, Restrepo M, Henderson SG, Topaloglu H (2010) Approximate dynamic programming for ambulance redeployment. INFORMS J Comput 22(2):266–281

Nunes LGN, de Carvalho SV, Rodrigues RCM (2009) Markov decision process applied to the control of hospital elective admissions. Artif Intell Med 47(2):159–171

Patrick J, Puterman ML, Queyranne M (2008) Dynamic multipriority patient scheduling for a diagnostic resource. Oper Res 56(6):1507–1525

Porter ME, Teisberg EO (2007) How physicians can change the future of health care. J Am Med Assoc 297(10):1103

Powell WB (2011) Approximate dynamic programming: solving the curses of dimensionality, 2nd edn., Wiley series in probability and statistics. Wiley, London

Schmid V (2012) Solving the dynamic ambulance relocation and dispatching problem using approximate dynamic programming. Eur J Oper Res 219(3):611–621

Schütz HJ, Kolisch R (2012) Approximate dynamic programming for capacity allocation in the service industry. Eur J Oper Res 218(1):239–250

Simao H, Powell WB (2009) Approximate dynamic programming for management of high-value spare parts. J Manuf Technol Manag 20(2):147–160

Topaloglu H, Powell WB (2006) Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems. INFORMS J Comput 18(1):31–42

Peter J. H. Hulshof studied Industrial Engineering and Management at the University of Twente, where he focused on operations management and logistics. He obtained his master's degree with a research on vehicle routing and order planning in the oil and gas sector. Peter focused his Ph.D on operations research in healthcare at the Center for Healthcare Operations Improvement and Research (CHOIR) at the University of Twente, under the supervision of prof. dr. ir. Erwin W. Hans and prof. dr. Richard J. Boucherie. During his Ph.D, he had a combined position at the University of Twente and the Reinier de Graaf hospital in the Netherlands.

Martijn R. K. Mes is an assistant professor within the department Industrial Engineering and Business Information Systems at the University of Twente, The Netherlands. He holds a master's degree in Applied Mathematics (2002) and did his Ph.D at the School of Management and Governance, University of Twente (2008). After finishing his Ph.D, Martijn did his postdoc at Princeton University, Department of Operations Research and Financial Engineering, where he did research on the topics of Ranking and Selection (R&S), Bayesian Global Optimization (BGO), and Optimal Learning. In general, Martijn's research involves healthcare logistics, sustainable logistics, multi-agent systems, dynamic vehicle routing problems, optimal learning, approximate dynamic programming (ADP), simulation optimization, discrete-event simulation, and simulation of logistics and healthcare systems.

Richard J. Boucherie received M.Sc degrees in 1988 in applied mathematics (stochastic operations research) and theoretical physics (statistical physics) from the Universiteit Leiden, and received the Ph.D degree in econometrics in 1992 for a thesis on Product-form in queueing networks from the Vrije Universiteit, Amsterdam. Following Post docs at INRIA Sophia Antipolis, CWI Amsterdam, and Universiteit van Amsterdam, since 2000 he is with the department of Applied Mathematics of the University of Twente, where he was appointed in 2003 as full professor of Stochastic Operations Research. His research interests are in queueing theory, Petri nets and random walks with application areas including wireless and sensor networks, healthcare, road traffic, and network intrusion detection and prevention. Richard is co-founder of the University of Twente research center CHOIR (Center for Healthcare Operations Improvement and Research) in the area of healthcare logistics.

Erwin W. Hans is a full professor Operations Management in Healthcare within the department Industrial Engineering and Business Information Systems at the University of Twente, The Netherlands.

He has a master's degree and Ph.D in Applied Mathematics. He is co-founder and co-chair of the Center for Healthcare Operations Improvement and Research (CHOIR), the Netherlands research center for OR/OM in healthcare research. He is director of the Industrial Engineering & Management undergraduate and graduate programs. His research focuses on Operations Management/Operations Research in the healthcare domain.