Thesis for the Degree of Master of Science

Maintenance Optimization in Stochastic Multi-component Systems

A Dynamic Programming Approach

Emil Gustafsson

Department of Mathematical Sciences
Division of Mathematics
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
Göteborg, August 2010

Abstract

The main purpose of the thesis is to investigate the possibilities of utilizing dynamic programming as a solution method for the stochastic opportunistic replacement problem (SORP). The objective in the SORP is to choose a maintenance policy that will minimize the expected total maintenance cost over a planning period for a system with stochastic component lives. Each component failure includes a maintenance occasion cost and a cost for replacing the failed component. A maintenance policy is a rule which determines whether it is beneficial to replace more than the necessary components given the system state.

A dynamic programming model is developed and some theoretical properties of the problem are presented. Three main methods are analyzed; an exact dynamic programming model, an infinite horizon simplification, and an approximate dynamic programming (ADP) model.

The size of the dynamic programming model becomes computationally intractable even for medium size problems, so the main results of the thesis is based on smaller test problems. For the problems where the exact dynamic programming model is applicable, the model gives superior results compared to other maintenance policies. That is, the total maintenance cost obtained by the dynamic programming model is lower than the other policies maintenance costs. The infinite horizon method gives suprisingly good results even though the simplification is substantional. The ADP model does not perform as well, but can on the other hand be used on larger problem instances.

Keywords: opportunistic replacement problem, dynamic programming, maintenance optimization, approximate dynamic programming.

Aknowledgements

I would like to thank to my supervisors Adam Wojciechowski and Michael Patricksson from the optimization research group at the Department of Mathematical Sciences at Chalmers University for their support. Without their help this project would not have been possible.

A final thank goes to my parents Lennart and Monica Gustafsson for always supporting me and inspiring me.

Emil Gustafsson, November 3, 2010

Contents

1 Introduction

Maintenance is needed on many types of systems and the performance of maintenance operations can in some cases be a large proportion of the total cost of running a system. One maintenance strategy this thesis will analyze is what is denoted as opportunistic maintenance. One type of opportunistic maintenance was studied by Dickman et al. [4], and Jorgensen and Radner [8], where a mathematical model was utilized to determine if doing more than the necessary maintenance might reduce the total cost over the maintenance period.

One model which utilizes opportunistic maintenance can be found in [1] and studies the opportunistic replacement problem (ORP). The ORP is a problem that considers a system of components which all have deterministic lives. When a component fails, a maintenance occasion cost is paid and the failed component has to be replaced. Because of the maintenance occasion cost included, it may prove beneficial to replace other components as well, because it might save an additional maintenance occasion cost in the future. The ORP can be solved by a mixed integer programming model and a replacement schedule for a whole planning period can be obtained. A large limitation the ORP is that it can not include stochastic components in the system. Even though it may seem like a simplification to assume some deterministic component lives, there are many systems where a deterministic life is justified. For example, aircraft engine components have regulations according to which they have to be replaced after a certain amount of flight hours. The probability of failure for these components before such a regulated life id very low. They can thus be considered as deterministic.

When adding uncertainty to an optimization problem, several difficulties emerge. The first problem is how to determine the objective in a problem which includes stochastic parameters. The most common solution is to formulate the problem as minimizing (maximizing) the expected cost (profit). This may not be preferred in some situations where the robustness of the solution has a large effect. For example, a solution with minimum expected cost might have a large variance that in the worst scenarios gives a very high maintenance cost. If we wish to prevent this, the objective might have to include the variance. Another problem that occurs when adding uncertainty is that the problem itself becomes much more complicated. To minimize an expectation of some ”easy” function may in some instances be extremely hard.

For the ORP, a very important feature occurs when adding uncertainty to the life of the components. When solving the ORP for deterministic lives, one gets an optimal replacement schedule which denotes exactly which components to replace at each timestep. If we include components with uncertain lifes, we can not predict how the system will behave (when the components fail) in the future. If we would like to have a replacement schedule for the whole planning period, we would have to construct one schedule for each possible outcome. Hence, the only decision we can make when a component failes is which components we should replace at that maintenance occasion. To decide which components to replace, we try to find the optimal decision which will minimize the cost we have to pay at the maintenance occasion plus the expected future maintenance cost.

To include systems with stochastic lives, the stochastic opportunistic replacement problem (SORP) has been formulated, see [10]. The SORP can be formulated as finding an optimal maintenance policy over a specified planning period. A maintenance policy is a function that tells us which components to replace at a maintenance occasion, depending on how long the planning period is and how old the components in the system are. To find such a maintenance policy, several solution mehtods can be applied. In [10], a simplified version of the SORP is formulated to find near-optimal policies by using a two-stage stochastic programming approach.

1.1 Purpose

The purpose of this thesis is to investigate the possibility of utilizing dynamic programming as a solution method for the SORP. Dynamic programming is a method for solving multi-stage decision problems and is closely related to the two-stage model in [10]. The notion of dynamic programming was first used by Richard Bellman when he was trying to solve complex multi-stage decision problems in the 1940's and he laid the foundation for the area by publishing *Dynamic Programming* [3] in 1959. The SORP has many properties that indicate that a dynamic programming approach to the problem should be interesting. The problem can be formulated as a Markov decision problem and it is a sequential decision problem where the objective is to find an optimal maintenance policy.

Approximate dynamic programming (ADP) is an area within dynamic programming which utilizes different approximation techniques to find near optimal policies and a large part of the thesis is concerned with finding approximations that might yield satisfactory policies.

1.2 Limitations

Dynamic programming often experience what is denoted as the curses of dimensionality. This referres to instances where the size of the state and decision spaces grows exponential and becomes computationally intractable even for medium size problems. The SORP has this property and we know that an exact dynamic programming model will not be applicable for larger instances of the SORP. Therefore, we have limited ourselves to the development and evaluation of the exact dynamic programming model only for smaller instances. ADP can, in some cases, overcome the problems regarding dimensionality. Therefore, when analyzing larger problem instances, the main focus will be the performance of the ADP algorithms.

Another aspect of the SORP is the question of discretization. In reality, the components in a stochastic system will not fail at certain discretized time steps but will fail according to some continuous probability distribution. To be able to utilize dynamic programming, we will assume a discretized time horizon and will not discuss how such discretizations effect the performance of the policies developed.

2 Dynamic Programming

Dynamic programming can be viewed as a method for solving multi-stage decision problems. The method breakes down complex problems into simpler subproblems. From an optimization point of view, dynamic programming simplifies a complex decision by dividing it into a sequence of simpler decision steps. Instead of trying to find optimal decisions for a whole time horizon, one can construct a subproblem for each timestep and solve each subproblem recursively. When utilizing dynamic programming, one views the system as being in different *states*. The optimization problem is to find the best possible *decision*, given that the system is in some specific state. When we make a decision, the system evolves in time. How the system evolves is modelled by a *transition function*, which is a function that takes the system from one state to another. The transition function gives the next state for the system, given an initial state, a decision, and some *exogenous information*. The exogenous information is something that ”arrives” to the system after we have made a decision. This information might depend on some stochastic process which has an impact on the system.

One simple example where dynamic programming can be utilized is stock trading. If we are interested in buying or selling a specific stock, we let the present value of the stock be the systems state. We have to utilize some optimization technique to determine whether we should buy or sell the stock, i.e. what decision we should take. After the decision is made, the stock price will move in some direction and the change in stock price is the exogenous information that arrives to the system.

When engineers and economists utilizes dynamic programming, they usually focus on problems with continuous states and decisions. In these kind of problems, one often views the decisions as some kind of controls and therefore, these problems are denoted as problems within *control theory*. In the fields of operations research and artificial intelligence, the problems often have discrete states and decisions and are called *Markov decision processes*. The Markov property in dynamic programming refers to systems where the state of the system captures the history of the system in such a way, that the only information needed to make a decision is the present state. Not all problems in dynamic programming can be formulated in such a way that they fulfill the Markov property. One example is trading with specific history dependent options. There exists a large amount of literature in the area of dynamic programming on problems that do not have the Markov Property, for example [13]. However, the problems we are analyzing do fulfill the Markov property, so in the future we are going to assume this property when dealing with dynamic programming.

In this thesis, we will use s_t to represent the state of the system at time t. The decision we make is denoted x_t and the exogenous information that arrives is denoted w_t. We use ϕ to represent the transition function and, with these notations, the system will evolve in time according to

$$s_{t+1} = \phi(s_t, x_t, w_t).$$

For each decision we make, a cost $c(s_t, x_t)$ has to be paid. If we first assume a deterministic system. the objective is to minimize the total cost over some planning period. If we assume that the system is in some state at time 0, and we have to make decisions for the time horizon $0, \ldots, T$, our problem is to

$$\begin{aligned} \text{minimize}_{x_0,\ldots,x_T} \quad & \left(\sum_{t=0}^{T} c(s_t, x_t)\right), \\ \text{subject to} \quad & s_{t+1} = \phi(s_t, x_t, w_t). \end{aligned}$$

This implies that that we need to choose decisions $x_0, \ldots, x_T$ such that the total cost we pay is minimized. This notation is correct as long as we assume that the system is deterministic, which is the same as having a model where the transition function does not depend on any exogenous information.

When we analyze a stochastic system, we assume that we are interested in minimizing the expected total cost we have to pay. A first guess would be to write down our objective as

$$\begin{aligned} \text{minimize}_{x_0,\ldots,x_T} \quad & \mathbb{E}\left(\sum_{t=0}^{T} c(s_t, x_t)\right), \\ \text{subject to} \quad & s_{t+1} = \phi(s_t, x_t, w_t). \end{aligned}$$

This formulation is not mathematically correct because the states s_t for $t = 1, \ldots, T$ are random variables. We cannot determine some specific decision x_t as the state at time t is random. Hence, we have to consider our decisions x_t as random variables.

This is where dynamic programming is needed for solving problems with stochastic exogenous information. Dynamic programming was developed in the 1940s by Richard Bellman to solve sequential decision problems. The biggest impact Richard Bellman did was to develop the Bellman equations. The Bellman equations break down a complex decision problem into a sequence of simpler subproblems. Each time step t yields a subproblem on the form

$$\begin{aligned} \min_{x_t} \quad & c(s_t, x_t) + \mathbb{E}\left(v_{t+1}(s_{t+1}) \mid s_t, x_t\right) \\ \text{s.t.} \quad & s_{t+1} = \phi(s_t, x_t, w_t). \end{aligned}$$

For each subproblem we are trying to find the best decision x_t that will minimize the cost we have to pay at time t plus the expected future cost. These subproblems and their connection with our "mathematically incorrect" objective function will be explained thoroughly in the following sections.

One common reason why dynamic programming is dismissed as a solution method for optimization problems are the problems regarding dimensionality. For a general stochastic dynamig programming model, three kinds of dimensionalities come up.

- ***The state space*** - The state variable $s_t = (s_{t1}, \ldots, s_{tI})$ has dimension I. If each s_{ti} can take on L possible values, then the number of states might be as large as L^I.
- ***The outcome space*** - The exogenous variable $w_t = (w_{t1}, \ldots, w_{tJ})$ has dimension J. If each w_{ti} can take on M possible values, then the number of outcomes might be as large as M^J.
- ***The decision space*** - The decision variable $x_t = (x_{t1}, \ldots, x_{tN})$ has dimension N. If each x_{ti} can take on K possible values, then the number of decisions might be as large as K^N.

2.1 The Optimality Equations

Assume a finite planning period from 0 to T. At time $0 \leq t \leq T$ the process is in state s_t and x_t is the decision variable. The feasible set of decisions, given that the system is in state s_t, will be denoted $\mathcal{X}_{s_t}$. We assume that we know the transition matrix $\mathbb{P}(s_{t+1} \,|\, s_t, x_t)$, which gives the probability that the system reaches state s_{t+1} at time $t+1$, given that the system is in state s_t and we make decision x_t at time t. This probability depends on the probability of the exogenous information w_t.

In this thesis, we assume that all the problems considered have the Markov property, i.e., the state s_t depends only on

- the state s_{t-1},
- the decision variable x_{t-1}, and
- the exogenous information w_{t-1}.

We denote the transformation function by ϕ, hence

$$s_{t+1} = \phi(s_t, x_t, w_t), \text{ for } 0 \leq t \leq T-1. \tag{1}$$

By taking decision x_t when we are in state s_t at time t, we have to pay the cost $c_t(s_t, x_t)$.

In a stochastic problem, the objective function can not be formulated in the same way as in a deterministic problem because of the exogenous information. Instead, we formulate the problem as finding the best policy for choosing x_t.

Definition 1 *A policy $\pi : \mathcal{S} \rightarrow \mathcal{X}$ is a mapping from the state space to the decision space that specifies decisions for all states in $\mathcal{S}$.*

As explained in the definition above, a policy π tells us what to do for all possible states. Let $x_t^\pi(s_t)$ be a decision made by policy π at time t when the system is in state s_t . By letting Π be the set of all possible decision policies, our problem can be formulated as

$$\min_{\pi \in \Pi} \mathbb{E} \left(\sum_{t=0}^{T} \gamma^t c_t(s_t, x_t^\pi(s_t)) \right), \tag{2}$$

where γ is a factor that can be included to take into account the time value of money. The minimum expected cost of being in state s_t is denoted as $v_t(s_t)$, which is a sum of the cost we have to pay at time t and the expected cost of the future, i.e.,

$$\begin{aligned} v_t(s_t) &= \min_{x_t \in \mathcal{X}_t} \left(c_t(s_t, x_t) + \gamma \mathbb{E}\left[v_{t+1}(s_{t+1}) \,|\, s_t, x_t \right] \right) \qquad (3) \\ \text{subject to} & \quad s_{t+1} = \phi(s_t, x_t, w_t). \end{aligned}$$

Equation 3 is often referred to as the expectation form of Bellman's equation. Another way of describing the minimun cost of being in state s_t is to express the expectation as a sum over the different states, i.e.,

$$\begin{aligned} v_t(s_t) &= \min_{x_t \in \mathcal{X}_t} \left(c_t(s_t, x_t) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s_{t+1} = s' \,|\, s_t, x_t) v_{t+1}(s') \right) \qquad (4) \\ \text{subject to} & \quad s_{t+1} = \phi(s_t, x_t, w_t). \end{aligned}$$

For simplicity, we let

$$p_{ss'}(x) = \mathbb{P}(s_{t+1} = s' \mid s_t = s, x_t = x)$$

be the probability that the state s will transform to state s' given that we make the decision x. If we are using a specific policy π to make the decisions, we write the probability as

$$p^{\pi}_{ss'} = \mathbb{P}(s_{t+1} = s' \mid s_t = s, x_t = x^{\pi}_t(s)).$$

By creating a matrix with elements $p^{\pi}_{ss'}$, we obtain a different probability matrix P^{π}_t for each polixy π. Let c^{π}_t and v_t be column vectors with elements $c(s, x^{\pi}_t(s))$ and $v_t(s)$ respectively. By using this notation, Bellman's equation is

$$v_t = \min_{\pi\in\Pi}(c^{\pi}_t + \gamma P^{\pi}_t v_{t+1}), \tag{5}$$

where the minimization is made componentwise. By solving these equations, an optimal policy π^* is obtained, and hence a decision $x^*_t(s_t)$ for all states.

The relationship between Bellman's equation and the objective function (2) is explained in Theorem 2.1.

Theorem 2.1 *Let $v_t(s_t)$ be the solution to Bellman's equation (4). Then*

$$v_t(s_t) = \min_{\pi\in\Pi} \mathbb{E}\left(\sum_{t'=t}^{T} \gamma^{t'} c_{t'}(s_{t'}, x^{\pi}_{t'}(s_{t'})) \mid s_t\right), \tag{6}$$

Proof See Apendix A.1

Theorem 2.1 tells us that to find the optimal policy π^* that will minimize the total cost from time t to T, given that the system is in state s_t, we can solve the Bellman equations for $v_t(s_t)$. The problem is that to solve such an equation, the values of v_{t+1} has to be known.

2.2 Pre- and Post-decision State Variables

In many practical applications, it is possible to break down the effect of the decision and the exogenous information. This implies that the original transition relation $s_{t+1} = \phi(s_t, x_t, w_t)$ can be divided into two steps

$$\begin{aligned} s^x_t &= \phi^x(s_t, x_t) \\ s_{t+1} &= \phi^w(s^x_t, w_t), \end{aligned}$$

where s_t is the state of the system just before we make a decision, and s^x_t is the state just after we have made decision x_t. s_t is referred to as the pre-decision state variable and s^x_t is referred to as the post-decision state variable. A schematic figure illustrating the difference between the two state variables can be seen in Figure 1.

As noted before, $v_t(s_t)$ is the cost (often denoted value) of being in state s_t just before we make a decision. Let $v^x_t(s^x_t)$ be the cost of being in state s^x_t just after we hade made a decision. The relationship between the cost of being in the pre- or post-decision state is

$$v^x_{t-1}(s^x_{t-1}) = \mathbb{E}[v_t(s_t) \mid s^x_{t-1}], \tag{7}$$

which says that the cost of being in the post-decision state s^x_{t-1} is equal to the expected cost of the next pre-decision state $v_t(s_t)$, given that decision x has been made. We can also write the pre-decision cost as a function of the post-decision cost by

$$v_t(s_t) = \min_{x_t\in\mathcal{X}_t} \left(c_t(s_t, x_t) + \gamma v^x_t(s^x_t)\right). \tag{8}$$

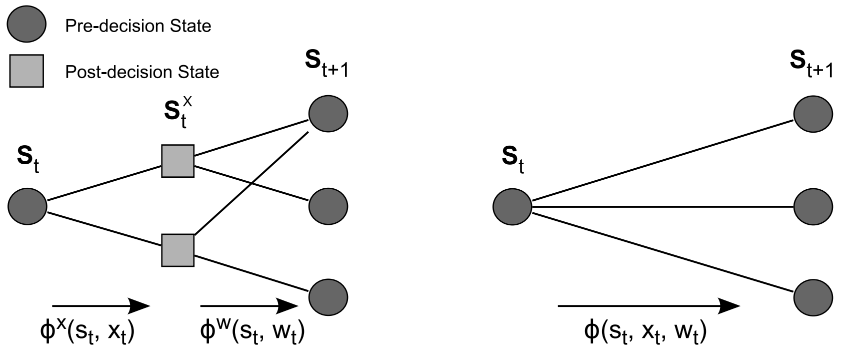


Figure 1: Schematic illustration of the difference between pre- and post-decision states. In the left figure, a state tree using post-decision variables is shown. In the right figure, a tree consisting of pre-decision variables only is shown.

If we combine the two equations, we obtain the optimality equations for the post-decision state variable

$$v_{t-1}^x(s_{t-1}^x) = \mathbb{E}\left[\min_{x_t \in \mathcal{X}_t} \left(c_t(s_t,x_t) + \gamma v_t^x(s_t^x)\right) \mid s_{t-1}^x\right]. \tag{9}$$

A clear difference between these optimality equations and the standard form of Bellman's equation is that the expectation is now outside the min operator. The advantage of using this form is that the optimization problem inside the expectation is a deterministic problem which can be solved by various optimization techniques. However, to evaluate $v_t^x(s_t^x)$, the expectation in equation (7) must be computed. This is often computationally intractable and in many cases, this is the part which is approximated.

2.3 Finite Horizon

Finite horizon problems are problems with a specific time horizon $T < \infty$. As described in section 2.1, the objective is to find a policy that solves

$$\min_{\pi \in \Pi} \mathbb{E}\left(\sum_{t=0}^{T} \gamma^t c(s_t, x_t^\pi(s_t)) \mid s_0\right).$$

By using Theorem 2.1, we know that this is equivalent to solving the Bellman equations for $v_0(s_0)$. If we know $v_T(s_T)$, we can obtain values of the functions v_t recursively from T back to 0, and thus find the optimal policy at time 0. In many cases, the cost at the terminal time T is set to $v_T(s_T) = 0$. This is referred to as backward dynamic programming and is presented in Algorithm 2.1.

```
Algorithm 2.1: BACKWARD DYNAMIC PROGRAMMING( )

  set terminal conditions $v_T(s_T)$ and let $t = T - 1$
  while $t \geq 0$
      for all $s_t \in S_t$
          solve
          $v_t(s_t) = \min_{x_t \in \mathcal{X}_t} \left( c_t(s_t,x_t) + \gamma \sum_{s' \in S_{t+1}} \mathbb{P}(s' \mid s_t, x_t) v_{t+1}(s') \right)$
          and let $x_t^*(s_t)$ be the optimal decision.
      end
      let $t = t - 1$
  end
  return $(x_0^*)_{t=0}^T$
```

To show how backward dynamic programming works, we look at at deterministic shortest path problem, see Figure 2. The objective is to find the optimal path from the start A to the goal E. To get to E, we have to pass through three different nodes, one in layer B, one in layer C and one in layer D. The numbers on each edge represent the length between the two connected nodes.

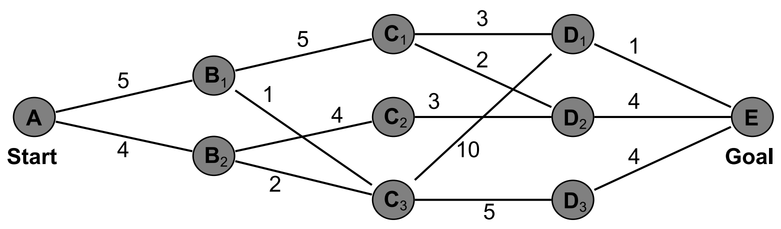


Figure 2: Shortest path problem.

To solve this problem with backward dynamic programming, we begin by finding the shortest path from all nodes in layer D to the end node E. Then we move to layer C and find the shortest path from all nodes in layer C to the goal by utlizing the information we obtained in the previous step. Next layer is B and is solved using the information about the shortest path from C to the goal. Finally, we go to our start node A and finds the shortest path. A schematic explanation of the algorithm can be found in Figure 3. By utilizing this algorithm, we see that the shortest path is to use the path $A \rightarrow B_1 \rightarrow C_1 \rightarrow D_1 \rightarrow E$, with a path length of 14.

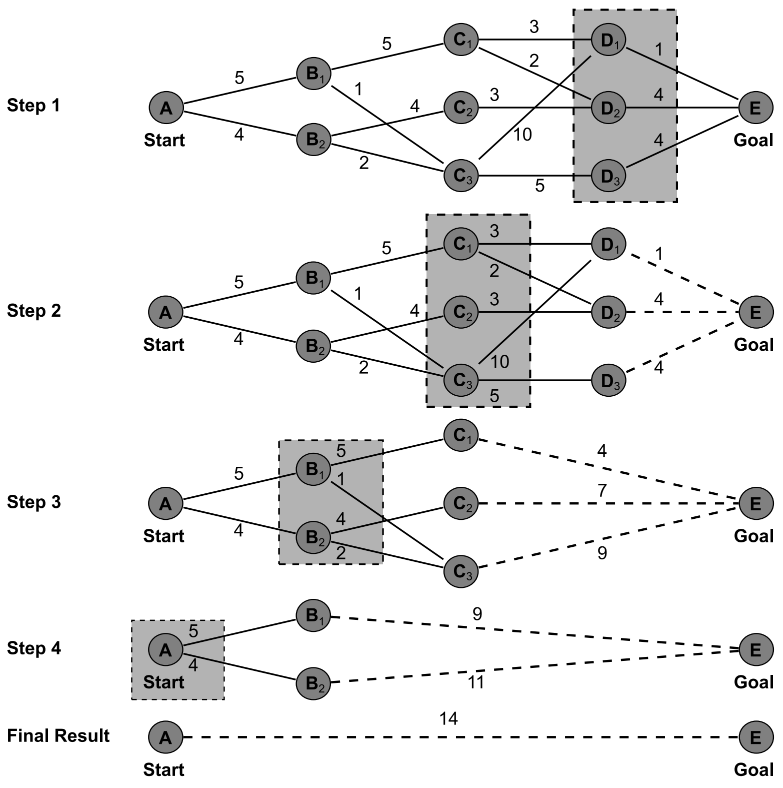


Figure 3: A schematic illustration of backward dynamic programming applied to a shortest path problem. The dashed lines represents the shortest path from a node to the goal.

2.4 Infinite Horizon

Infinite time horizon problems are problems where the parameters of the underlying processes do not vary over time.

By letting the time horizon $T \to \infty$, and $v(s) = \lim_{t\to\infty} v_t(s_s)$ (assuming the limit exists), we obtain the Bellman equation for an infinite horizon problem,

$$v(s) = \min_{x \in \mathcal{X}_s} \left(c(s,x) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s,x) v(s') \right). \tag{10}$$

The main difference between the Bellman equations for finite and infinite horizon problems is that the infinite horizon problem equation does not contain a time dimension. In the infinite horizon case we define the total discounted cost, given that we follow a specific stationary policy π, to be

$$v^\pi(s) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t c(s_t, x_t^\pi(s_t)) \mid s_0 = s \right). \tag{11}$$

The optimality equations for the infinite horizon case is presented in Theorem 2.2

Theorem 2.2 *Let $v(s)$ be the solution to Bellman's equation for state $s \in \mathcal{S}$. Then*

$$\begin{aligned} v(s) &= \min_{\pi \in \Pi} v^\pi(s) \\ &= \min_{\pi \in \Pi} \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t c_t(s_t, x_t^\pi(s_t)) \mid s_0 = s \right) \end{aligned}$$

Proof See [11]

As in section 2.3, let $p_{ss'}(x)$ be the probability that the system is transformed to state s', given that it is in state s and we make decision x. For a given policy π, let the transition matrix P^π consist of the probabilities $p^\pi_{ss'}$. Let c^π and v be column vectors with elements $c(s, x^\pi(s))$ and $v(s)$ respectively. For a given policy π, we define the operator $\mathcal{T}^\pi : \mathcal{V} \to \mathcal{V}$ as

$$\mathcal{T}^\pi v = c^\pi + \gamma P^\pi v,$$

where $\mathcal{V}$ is the set of bounded, real-valued functions on $\mathcal{S}$. Also, we define the operator $\mathcal{T} : \mathcal{V} \to \mathcal{V}$ as

$$\begin{aligned} \mathcal{T}v &= \min_{\pi \in \Pi} \mathcal{T}^\pi v \\ &= \min_{\pi \in \Pi} (c^\pi + \gamma P^\pi v). \end{aligned}$$

where the minimization is carried out componentwise.

Definition 2 *Let $\mathcal{V}$ be ths space of bounded, real-valued functions. The* ***norm*** *of $\mathcal{V}$ is then*

$$||v|| = \sup_{s \in \mathcal{S}} v(s).$$

We can replace the "sup" with a "max" if the state space is finite.

Definition 3 *A mapping $\mathcal{M} : \mathcal{V} \to \mathcal{V}$ is a* ***contraction mapping*** *if there exists a $0 \le \alpha < 1$ such that*

$$||\mathcal{M}v - \mathcal{M}u|| \le \alpha ||u - v||, \text{ for } u,v \in \mathcal{V}.$$

Theorem 2.3 (Banach Fixed-Point Theorem) *Let $\mathcal{V}$ be a Banach space, and let $\mathcal{M} : \mathcal{V} \to \mathcal{V}$ be a contraction mapping. Then*

$$\begin{aligned} &(a) \text{there exist a unique } v^* \in \mathcal{V} \text{ such that } \mathcal{M}v^* = v^* \text{and,} \\ &(b) \text{for an arbitrary } v^0 \in \mathcal{V}, \text{ the sequence } v^n \text{ defined by } v^{n+1} = \mathcal{M}v^n \text{ converges to } v^*. \end{aligned}$$

By analyzing the operators we just defined, we can show the following proposition.

Proposition 2.4 *If $0 \le \gamma < 1$, then $\mathcal{T}$ is a contraction mapping on $\mathcal{V}$.*

Proof Let $u, v \in \mathcal{V}$ and assume that $\mathcal{T}v(s) \ge \mathcal{T}u(s)$ for a state s. Let

$$x_s^*(v) = \arg\max_{x \in \mathcal{X}} \left(c(s,x) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s, x) v(s') \right),$$

where we assume that the solution exists. Then

$$\begin{aligned}
0 &\leq \mathcal{T}v(s) - \mathcal{T}u(s) \\
&= c(s, x_s^*(v)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \,|\, s, x_s^*(v))v(s') \\
&\quad - \left(c(s, x_s^*(u)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \,|\, s, x_s^*(u))u(s') \right) \\
&\leq c(s, x_s^*(v)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \,|\, s, x_s^*(v))v(s') \\
&\quad - \left(c(s, x_s^*(v)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \,|\, s, x_s^*(v))u(s') \right) \\
&= \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \,|\, s, x_s^*(v))(v(s') - u(s')) \\
&\leq \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \,|\, s, x_s^*(v))||v - u|| \\
&= \gamma ||v - u|| \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \,|\, s, x_s^*(v)) \\
&= \gamma ||v - u||.
\end{aligned}$$

By using the same reasoning if $\mathcal{T}v(s) \leq \mathcal{T}u(s)$, we get that

$$|\mathcal{T}v(s) - \mathcal{T}u(s)| \leq \gamma ||v - u||,$$

for all states $s \in \mathcal{S}$. From the definition of our norm, we conclude that

$$\begin{aligned}
\sup_{s \in \mathcal{S}} |\mathcal{T}v(s) - \mathcal{T}u(s)| &= ||\mathcal{T}v - \mathcal{T}u|| \\
&\leq \gamma ||u - v||.
\end{aligned}$$

□

Now Proposition 2.4 implies that Banach's fixed point theorem can be applied to the operator in Bellman's equation ans implies that

$$v = \mathcal{T}v$$

has an unique solution v^*. Furthermore, this unique solution is the minimal cost function (or optimal cost-to-go function)

$$v^* = \min_{\pi \in \Pi} v^\pi.$$

If v^* is found, the optimal decision policy π^* can be generated according to

$$x^{\pi^*}(s) = \arg\min_{x \in \mathcal{X}} \left(c(s, x) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \,|\, s, x) v^*(s') \right).$$

2.4.1 Value Iteration

Banach's fixed point theorem also tells us that if we start with any initial guess v^0 and then utilize the operator $\mathcal{T}$ iteratively, we will eventually converge to the true value of v^*. This method is often referred to as value iteration.

The value iteration algorithm for infinite horizon problems can be viewed as the method that corresponds to backward dynamic programming for finite horizon problems. The difference is that we are analyzing a infinite horizon, which implies that we have to have a stopping criteria for the algorithm. If we terminate the algorithm when

$$||v^n - v^{n-1}|| < \epsilon(1-\gamma)/2\gamma, \tag{12}$$

a bound on the error between the the solution v^n and the optimal value v^* is obtained from Theorem 2.5

Theorem 2.5 *If we apply the value iteration algorithm with stopping parameter ϵ according to (12), and the algorithm terminates at iteration n with value function v^{n+1}, then*

$$||v^{n+1} - v^*|| \leq \epsilon/2.$$

Let π^ϵ be the greedy policy determined by v^{n+1}, then

$$||v^{\pi^\epsilon} - v^*|| \leq \epsilon.$$

Proof See [11].

There are many versions of the value iteration method in the litterature but for larg-scale problems, the algorithm is often computationally intractable. At each step of the algorithm we loop over all possible states. For many problems, the size of the state space is too large for such a procedure to be possible.

2.4.2 Policy Iteration

By choosing a policy π, we would find a cost-to-go function v^π by solving

$$\begin{aligned} v^\pi &= T^\pi v^\pi \\ &= c^\pi + \gamma P^\pi v^\pi. \end{aligned}$$

If we assume that $0 \leq \gamma < 1$, this equation system has a unique solution

$$(I - \gamma P^\pi)^{-1} v^\pi = c^\pi,$$

where I is the identity matrix. What this tells us is that if we choose a policy π, we obtain the cost of that particular policy by solving the linear system (2.4.2). We then update our policy by using a greedy method with the obtained values v^π. Repeating this procedure leads to a convergence towards the optimal policy. The policy iteration algorithm has fast convergence but clearly it is a computationally hard problem if the state space is large because solving the linear system (2.4.2) becomes computationally intractable.

2.4.3 Linear Programming for Infinite Horizon Problems

As we have earlier in this section, Bellman's equations for a infinite horizon problem can be written on the compact form

$$v = \mathcal{T} v,$$

where v is the vector of the values for all states. We first provide a theorem which states some relationships between a vector v and the optimal cost-to-go vector v^*.

Theorem 2.6 *For a vector $v \in \mathcal{V}$,*

$$\begin{array}{lll} (a) & \textit{If } v \textit{ satisfies } v \geq \mathcal{T}v, & \textit{then } v \geq v^*, \\ (b) & \textit{If } v \textit{ satisfies } v \leq \mathcal{T}v, & \textit{then } v \leq v^*, \\ (c) & \textit{If } v \textit{ satisfies } v = \mathcal{T}v, & \textit{then } v \textit{ is the unique solution to } v = \mathcal{T}v \textit{ and } v = v^*. \end{array}$$

Proof See [12]. □

Theorem 2.6 tells us that if

$$v \leq \mathcal{T}v,$$

then v is a lower bound on the optimal solution v^*. Consider the problem

$$\begin{array}{ll} \text{maximize} & \beta^T v, \\ \text{subject to} & \mathcal{T}v \geq v. \end{array} \tag{13}$$

We refer to the vector β as a vector of *state-relevance weights*. By using Theorem 2.6, we can conclude that any feasible v in (13) satisfies $v \leq v^*$. So for any $\beta \geq 0$, v^* is the unique solution to (13).

Since $\mathcal{T}$ is a nonlinear operator, the constructed problem is not linear. But the problem can be reformulated as a linear program. First we can state (13) equivalently as

$$\begin{array}{ll} \text{maximize} & \beta^T v, \\ \text{subject to} & (\mathcal{T}v)(s) \geq v(s), \quad \forall s \in \mathcal{S}. \end{array} \tag{14}$$

Each constraint in problem (14) is equivalent to the set of constraints

$$c(s,x) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s, x) v(s') \geq v(s), \quad \forall x \in \mathcal{X}_s.$$

Now problem (13) can be rewritten as a linear program

$$\begin{array}{ll} \text{maximize} & \beta^T v \\ \text{subject to} & c(s,x) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s, x) v(s') \geq v(s), \quad \forall s \in \mathcal{S}, x \in \mathcal{X}_s. \end{array} \tag{15}$$

This means that the cost-to-go function v can be found by solving the linear program (15) with any weights β. Even though it is a linear program, since the number of constraints is $|\mathcal{S}| \times |\mathcal{X}|$, it still can be very difficult to solve.

3 Approximate Dynamic Programming

Because of the problems with dimensionality (see section 2) when using dynamic programming, the solution method is often computationally intractable. To be able to solve larger problems with dynamic programming, approximations have to be made. One method is what often is denoted as *approximate dynamic programming*.

The basic idea behind approximate dynamic programming is that one approximates the cost of being in different states and then iteratively updates the approximation. Instead of stepping backwards in time as with exact dynamic programming, we will step forward by simulating the process of the system. The problem we face when stepping forward in time is that the cost of being in different states in the future are unknown. These costs have to be approximated iteratively.

3.1 Finite Horizon

To find an optimal decision at time t, given that the system is in state s_t, Bellman's equation has to be solved, i.e.

$$x_t = \arg\max_{x\in\mathcal{X}} \left(c_t(s_t, x) + \gamma\mathbb{E}[v_{t+1}(s_{t+1}) \mid s_t]\right). \tag{16}$$

Because the future costs v_{t+1} are unknown, we begin by approximating them with some initial guess V_t^0 for all t. If no clear approximation for the costs exist, one often uses $V_t^0(s_t) = 0$ for all states. These costs will be updated iteratively. Let V_t^{n-1} be the cost approximation after $n-1$ iterations. In iteration n we use the approximated costs V_t^{n-1} instead of the unknown costs v_t, i.e. we will choose decision x_t^n by solving

$$\begin{aligned}
\hat{v}_t^n(s_t^n) &= \max_{x\in\mathcal{X}} \left(c_t(s_t^n, x) + \gamma\mathbb{E}[V_{t+1}^{n-1}(s_{t+1}) \mid s_t^n]\right) \\
&= \max_{x\in\mathcal{X}} \left(c_t(s_t^n, x) + \gamma \sum_{s'\in\mathcal{S}} \mathbb{P}(s' \mid s_t^n, x) V_{t+1}^{n-1}(s_{t+1})\right).
\end{aligned}$$

By solving the equation above, we obtain an approximation of the cost of being in state s_t^n. Instead of letting this estimate be our new approximation, we use a smoothing operation to update our estimate, i.e.,

$$V_t^n(s_t^n) = (1-\alpha_{n-1})V_t^{n-1}(s_t^n) + \alpha_{n-1}\hat{v}_t^n(s_t^n).$$

The next step is to move forward in time. We do this by first taking the decision x_t^n that solves the optimization problem in iteration n. Then, we pick a sample realization of exogenous information at random. The decision taken, x_t^n, and the sample realization, w_t^n, will decide which state the system visits in the next time step through the transition function $s_{t+1}^n = \phi(s_t^n, x_t^n, w_t^n)$. In the next time step, we solve the new optimization problem

$$\hat{v}_{t+1}^n(s_{t+1}^n) = \max_{x\in\mathcal{X}} \left(c_{t+1}(s_{t+1}^n, x) + \gamma \sum_{s'\in\mathcal{S}} \mathbb{P}(s' \mid s_{t+1}^n, x) V_{t+2}^{n-1}(s_{t+2})\right)$$

and update the value $V_{t+1}^n(s_{t+1}^n)$ as explained above. An approximate dynamic programming algorithm can be seen in Algorithm 3.1.

Algorithm 3.1: APPROXIMATE DYNAMIC PROGRAMMING( )

```
set initial approximations $V_t^0(s_t)$ for all states $s_t$
    initial state $s_0^1$.
while $n \le N$, do
    for $t = 0,1,\ldots,T$
        Let
            $\hat{v}_t := \min_{x_t \in \mathcal{X}_t} \left( c_t(s_t^n, x_t) + \gamma \sum_{s' \in S} \mathbb{P}(s' \mid s_t^n, x_t) V_{t+1}^{n-1}(s') \right)$
        and let $x_t^n$ be the optimal decision.

        Let
            $V_t^n(s_t^n) := (1 - \alpha_{n-1}) V_t^{n-1}(s_t^n) + \alpha_{n-1} \hat{v}_t$

        Draw a sample $w_t^n$ and let
            $s_{t+1}^n := \phi\left(s_t^n, x_t^n, w_t^n\right)$
    end
    Let $n = n + 1$
end
return $x_0^n$
```

3.1.1 Using Post-decision State Variables

By using post-decision state variables, the approximate dynamic programming algorithm is simplified. The basic idea is that instead of using the pre-decision state costs $v_t(s_t)$, we will use the post-decision costs $v_t^x(s_t^x)$, which to simplify notation we will write as $v_t(s_t^x)$. As explained in section 2.2, Bellman's equation corresponding to the post-decision state variables can be written as

$$v_{t-1}(s_{t-1}^x) = \mathbb{E}\left[\min_{x \in \mathcal{X}_t} \left(c_t(s_t,x) + \gamma v_t(s_t^x) \right) \mid s_{t-1}^x \right].$$

As mention before, the main differences is that the optimization problem is inside the expectation and that the optimization problem is deterministic. However, to use approximate dynamic programming with the post-decision state variables, we still have to approximate the post-decision costs $v_t(s_t^x)$ iteratively. As before, we start with some initial guess $V_t^0(s_t^x)$ for all states and times $t = 0, \ldots, T$.

Assume that we are in iteration n and at time $t-1$ the post-decision state is $s_{t-1}^{x,n}$. By drawing a sample w_{t-1}^n, we can compute the next pre-decision state by

$$s_t^n = \phi^w\left(s_{t-1}^{x,n}, w_{t-1}^n\right).$$

Now at time t, we have to solve the optimization problem

$$\hat{v}_t^n = \min_{x \in \mathcal{X}_t} \left(c_t(s_t^n, x) + \gamma V_t^{n-1}\left(\phi^x(s_t^n, x)\right) \right),$$

where V_t^{n-1} are the approximations for the post-decision state costs after $n-1$ iterations. The most important property of this problem is that it is deterministic. We now use $\hat{v}_t^n$ to update our estimate of the cost of being in post-decision state $s_{t-1}^{x,n}$ by

$$V_{t-1}^n(s_{t-1}^{x,n}) = (1 - \alpha_{n-1}) V_{t-1}^{n-1}(s_{t-1}^{x,n}) + \alpha_{n-1} \hat{v}_t^n. \tag{17}$$

The exact value of being in post-decision state s^x_{t-1} is

$$v_{t-1}(s^x_{t-1}) = \mathbb{E}\left[v_t(s_t) \,|\, s^x_{t-1}\right],$$

but, by using the smoothing operator when doing the update in equation (17), we approximate the expectation after a large number of iterations. The ADP algorithm for the post-decision state variables is presented in Algorithm 3.2

```
Algorithm 3.2: ADP WITH POST-DECISION STATE VARIABLES( )

  set initial approximations $V_t^0(s_t^x)$ for all states $s_t^x$
       initial state $s_0^1$.
  while $n \leq N$, do
      for $t = 0,1,\ldots,T$

          Let
              $\hat{v}_t := \min_{x_t \in \mathcal{X}_t} \left(c_t(s_t^n, x_t) + \gamma V_t^{n-1}(\phi^x(s_t^n, x))\right)$
          Let $x_t^n$ be the optimal decision.

          if $t > 0$, let
              $V_{t-1}^n(s_t^{x,n}) := (1-\alpha_{n-1})V_{t-1}^{n-1}(s_{t-1}^{x,n}) + \alpha_{n-1}\hat{v}_t$
          end

          Let
              $s_t^{x,n} := \phi^x\left(s_t^n, x_t^n\right)$

          Draw a sample $w_t^n$ and let
              $s_{t+1}^n := \phi^w\left(s_t^{x,n}, w_t^n\right)$
      end
      Let $n = n + 1$
  end
  return $x_0^n$
```

3.1.2 Approximating the Value Function

The most common problem in dynamic programming is that the state space is too large to handle. This problem is not solved by the algorithms above because we still have to find approximations for the values $v(s)$ for all $s \in \mathcal{S}$. To overcome this problem, we create a function $\hat{V}_t$ that will give an approximate value $\hat{V}_t(s_t)$ for all $s_t \in \mathcal{S}$. The difference between this approximation and V_t used above, is that V_t is a vector containing the approximate values for all different states, whereas $\hat{V}_t$ is a function that will produce a value $\hat{V}_t(s_t)$, given a state s_t. The computational advantage of using such an approximation is that when using an ADP algorithm, we do not have to update the values for all states. We only have to update the parameters of the function $\hat{V}_t$. The most common way to approximate the value functions is on the form

$$\hat{V}_t(s_t) = \sum_{k \in \mathcal{K}} \phi_k(s_t)\theta_k, \tag{18}$$

where ϕ_k, $k \in \mathcal{K}$ is a set of basis functions. When doing an iteration in an ADP algorithm, we update the parameters θ by an updating rule.

3.2 Infinite Horizon

As we have described in section 2.4.3, the infinite horizon problem can be formulated as a linear program

$$\begin{aligned} \max \quad & \beta^T v && (19)\\ s.t. \quad & c(s,x) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s, x) v(s') \geq v(s), \quad \forall s \in \mathcal{S}, x \in \mathcal{X}_s, \end{aligned}$$

where v is a vector of the cost-to-go values $v(s)$ for all states $s \in \mathcal{S}$. Because the state space in many cases is too large to handle, we use the same principles of apprixomations as in section 3.1. We begin by constructing some basis vectors $\phi_1, \ldots, \phi_K$, depending on s, and define the matrix

$$\Phi = \begin{bmatrix} | & & | \\ \phi_1 & \vdots & \phi_K \\ | & & | \end{bmatrix},$$

where each row corresponds to the basis functions for a specific state. The objective is to find a weight vector $\tilde{r} \in \mathbb{R}^K$ such that $\Phi \tilde{r}$ is a close approximation to v^*. To find such a weight vector, we solve

$$\begin{aligned} \max \quad & \beta^T \Phi r && (20)\\ s.t. \quad & \mathcal{T} \Phi r \geq \Phi r. \end{aligned}$$

(21) can be reformulated as an LP, to

$$\begin{aligned} \text{maximize} \quad & \beta^T \Phi r && (21)\\ \text{subject to} \quad & c(s,x) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s, x)(\Phi r)(s') \geq (\Phi r)(s), \quad s \in \mathcal{S}, x \in \mathcal{X}_s. \end{aligned}$$

After finding an optimal weight vector, we construct a policy $\pi^{\tilde{r}}$ by letting

$$\pi^{\tilde{r}}(s) = \arg\min_{x \in \mathcal{X}_s} \left(c(s,x) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s, x)(\Phi \tilde{r})(s') \right). \qquad (22)$$

By using the function approximation, the number of variables is reduced from $|\mathcal{S}|$ to K, but the number of constraints is still as large as before. Fortunately, for many problems a large proportion of the constraints are inactive. Numerical studies on different kinds of dynamic programming models have been performed and some theoretical results can be found in the litterature (see for instance [5], [6] regarding constraint sampling). Instead of creating constraints for every state-action pair, one can sample constraints by moving forward in time and adding the new constraints to the LP.

One difference between this approximate LP and the exact LP is that the state weights β will effect the quality of the solution. In the exact LP, every feasible solution to the LP produces the optimal cost-to-go values for any β. In the approximate LP, one can show that the choice of β actually effects the error bounds $||\Phi \tilde{r} - v^*||$,)see [6]).

4 Maintenance Optimization Problem

Consider a system consisting of a set of components $\mathcal{N} = \{1,\ldots,N\}$. Each component has a failure distribution according to the distribution function $F_i(a) = \mathbb{P}(\tau_i \leq a)$, where τ_i is the time of failure for component $i \in \mathcal{N}$. For simplicity, we denote $p_i(a) = \mathbb{P}(\tau_i \leq a+1 | \tau_i > a)$. If there exist deterministic components with life lenghts T_i, we set $p_i(T_i - 1) = 1$ and $p_i(a) = 0$ for $a \neq T_i - 1$. We assume some time discretization and denote $\mathcal{T} = \{0,\ldots,T\}$

If a component $i \in \mathcal{N}$ is broken at any given time step, that component has to be replaced and a cost of c_i has to be paid. Also, for each time we replace any component, an additional service cost of d has to be paid. Our objective is to find an optimal maintenance policy which will minimize the total cost for the maintenance period from time 0 to time T, i.e. we are trying to find decisions that minimizes

$$\text{(cost today)} + \text{(expected future cost)}$$

4.1 Variables

The state variables we will use are denoted by vectors

$$s_t = (s_{t1},\ldots,s_{tN}), \tag{23}$$

where s_{ti} represents the age of component $i \in \mathcal{N}$ at time $t \in \mathcal{T}$. If a component is broken, the age of that component is set to ∞. The decision taken at time t is which components to replace and is denoted by

$$x_t = (x_{t1},\ldots,x_{tN}), \tag{24}$$

where

$$x_{ti} = \begin{cases} 1 & \text{if component } i \in \mathcal{N} \text{ is replaced at time } t \in \mathcal{T} \\ 0 & \text{otherwise} \end{cases}$$

For each state s_t, the set of all allowed decisions is denoted by

$$\mathcal{X}_{s_t} = \begin{cases} \{0\}, & \text{if } s_t < \infty, \\ \{(x_{t1},\ldots,x_{tn}) \ : \ x_{it} = 1 \,\text{for}\, i \in \mathcal{B}\} & \text{if } s_{ti} = \infty \text{ for } i \in \mathcal{BN} \end{cases}$$

which implies that if no component has failed, maintenance is not performed. If, however, a subset $\mathcal{B} \subseteq \mathcal{N}$ of components are broken, those components have to be replaced. However, we have the option to replace yet non failed components.

After we have taken a decision x_t, the system recieves exogenous information about which components that fail between the current time step t and the next $t+1$. This information is denoted by

$$w_t = (w_{t1},\ldots,w_{tN}), \tag{25}$$

where

$$w_{ti} = \begin{cases} 1, & \text{if component } i \in \mathcal{N} \text{ fails between time } t \text{ and } t+1, \\ 0, & \text{otherwise.} \end{cases}$$

4.2 Transformation Function

Given a state s_t, a decision x_t, and exogenous information w_t, the system will be transformed to state s_{t+1} according to

$$(s_{t+1,1},\ldots,s_{t+1,N}) = (\phi_1(s_{t1},x_{t1},w_{t1}),\ldots,\phi_N(s_{tN},x_{tN},w_{tN})), \tag{26}$$

where

$$\phi_i(s_{ti}, x_{ti}, w_{ti}) = \begin{cases} s_{ti}+1, & \text{if } x_{ti}=0 \text{ and } w_{ti}=0, \\ 1, & \text{if } x_{ti}=1 \text{ and } w_{ti}=0, \\ \infty, & \text{otherwise} \end{cases}$$

In the remainder of this thesis we will write the transition relation in short as $s_{t+1} = \phi(s_t, x_t, w_t)$.

4.3 Cost Function

The cost of making decision x_t at time t is only dependent on the decision x_t according to is

$$c(x_t) = \begin{cases} 0, & \text{if } x_t = 0, \\ d + c^T x_t, & \text{otherwise,} \end{cases}$$

which says that if we decide to replace a set of components, we have to pay a service cost d plus the cost of the individual components. If we do not replace anything, the cost is 0.

4.4 Transition Probabilities

The exogenous information w_t depends on both s_t and x_t. We assume that w_{ti} only depends on s_{ti} and x_{ti}, which is to say that the probability distributions of the components are independent of each other. Assume that component $i \in \mathcal{N}$ has age s_{ti} and we take decision x_{ti}, then the probability that component i will fail between time step t and time step $t+1$ is

$$\mathbb{P}\left(w_{it} = 1\right) = \begin{cases} \mathbb{P}(\tau_i \leq s_{ti}+1 \,|\, \tau_i > s_{ti}), & \text{if } x_{ti} = 0, \\ \mathbb{P}(\tau_i \leq 1 \,|\, \tau_i > 0), & \text{if } x_{ti} = 1 \end{cases},$$

which says that if we do not replace the component, the probability that it will fail before time step $t+1$ is $p_i(s_{ti})$. If we do replace the component, the probability that it will fail is $p_i(0)$. This can be written in a more compact form as

$$\mathbb{P}(w_{ti} = 1) = p_i\left((1 - x_{ti})s_{ti}\right). \tag{27}$$

We procede by analyzing the transition probability between states. Assume that a specific exogenous information w_t arrives such that $w_{ti} = 1$ for $i \in \mathcal{B} \subseteq \mathcal{N}$. The probability for such exogenous information, given that the system is in state s_t and we take decision x_t, is

$$\prod_{i \in \mathcal{B}} p_i\left((1 - x_{ti})s_{ti}\right) \prod_{i' \in \mathcal{N} \setminus \mathcal{B}} \left(1 - p_{i'}\left((1 - x_{ti'})s_{ti'}\right)\right).$$

For this exogenous information, the next state the system will occupy is

$$s_{t+1,i} = \begin{cases} \infty, & \text{if } i \in \mathcal{B}, \\ (1 - x_{ti})s_{ti} + 1, & \text{otherwise.} \end{cases}$$

Assume that the system is in state s_t and we take decision x_t. Define the set $\mathcal{C} = \{\rangle \in \mathcal{N} \mid \S_{\sqcup\rangle} = \infty\}$ and assume that a set $\mathcal{B} \in \mathcal{N}$ of components fail between time step t and time step $t+1$. We create the following sets

$\mathcal{A}_1$	$= (\mathcal{N} \setminus \mathcal{C}) \cap \mathcal{B}$,	the set of componets that were not replaced and failed,
$\mathcal{A}_2$	$= (\mathcal{N} \setminus \mathcal{C}) \setminus \mathcal{B}$,	the set of componets that were not replaced and did not fail,
$\mathcal{A}_3$	$= \mathcal{C} \cap \mathcal{B}$,	the set of componets that were replaced and failed,
$\mathcal{A}_4$	$= \mathcal{C} \setminus \mathcal{B}$,	the set of componets that were replaced and did not fail.

Using these sets, we can express the next state as

$$s_{t+1,i} = \begin{cases} \infty, & \text{for } i \in \mathcal{A}_1 \cup \mathcal{A}_3 = \mathcal{B} \\ 1, & \text{for } i \in \mathcal{A}_4 \\ s_{ti} + 1, & \text{for } i \in \mathcal{A}_2 \end{cases}$$

The transition probability between states s_t and s_{t+1} given decision x_t is the probability of recieving exogenous information w_t with $w_{ti} = 1$ for $i \in \mathcal{B} \subseteq \mathcal{N}$, given s_t and x_t, which is

$$\mathbb{P}(s_{t+1} \mid s_t, x_t) = \prod_{i \in \mathcal{A}_1} p_i(s_{ti}) \prod_{j \in \mathcal{A}_2} (1 - p_j(s_{tj})) \prod_{k \in \mathcal{A}_3} p_k(0) \prod_{l \in \mathcal{A}_4} (1 - p_k(0)) \,.$$

We will assume that the components in the system have lives distributed according to the Weibull distribution, that is

$$p_i(s) = \mathbb{P}(\tau_i \leq s + 1 \mid \tau_i > s) = \frac{\int\limits_s^{s+1} f(x)dx}{\int\limits_s^{\infty} f(x)dx},$$

where f is the probability distribution function for the Weibull distribution, given by

$$f(x; \alpha, \beta) = \begin{cases} \frac{\beta}{\alpha} \left(\frac{x}{\alpha}\right)^{\beta - 1} \exp\left(-\left(\frac{x}{\alpha}\right)^{\beta}\right), & \text{if } x \geq 0, \\ 0, & \text{otherwise.} \end{cases}$$

For $\beta \geq 1$, the Weibull distribution has a non-decreasing hazard rate. In the replacement problem, we will assume that $\beta \geq 1$, which means that $p_i(s^+) \geq p_i(s^-)$ for $s^+ \geq s^-$ for all $i \in \mathcal{N}$.

4.5 Objective

Our objective is to minimize the total cost of maintenance over a finite time horizon. Because the system we are performing maintenance on is stochastic, we cannot construct a maintenance schedule for the future. What we can do is determine what the best possible decision is today that will minimize the sum of the cost today and the expected future cost. We assume that the system is in state s_0 at time $t = 0$. Mathematically, we are trying to find the best policy π that solves

$$\min_{\pi \in \Pi} \mathbb{E} \left(\sum_{t=0}^{T} \gamma^t c(x_t) \mid s_0 \right).$$

When studying a finite time horizon, we will use $\gamma = 1$. We showed in section 2.1 that this is equivalent to solving Bellman's equations, that is,

$$v_t(s_t) = \min_{x \in \mathcal{S}_{s_t}} \left(c(x) + \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s_t, x) v(s') \right), \quad \forall s \in \mathcal{S}, \quad \text{for } t = T, T-1, \ldots, 0.$$

We use $v_t(s)$ to denote the minimal expected maintenance cost for the time horizon $t, t+1, \ldots, T$, given that the system is in state s at time t. It seems reasonable to expect that v_t is an increasing function because starting out with older components should yield a larger expected maintenance cost than staring out with new components. The following theorem proves this fact.

Theorem 4.1 *Let v_t be defined as the solution to Bellman's equation for the replacement problem, i.e.*

$$v_t(s_t) = \min_{x_t \in \mathcal{X}_t} \left(c(x) + \sum_{s' \in S} \mathbb{P}(s' \mid s_t, x_t) v_{t+1}(s') \right)$$

where $c(x) = 0$ if $x = 0$ and $c(x) = d + c^T x$ otherwise. Let s^+ and s^- be states such that $s^+ \geq s^-$. Then

$$v_t(s^+) \geq v_t(s^-), \text{ for } t = 0, \dots, T$$

Proof The proof is done by induction. We first analyze the case when $t = T$. Assume a set of failed components $\mathcal{K}^+ = \{i \in \mathcal{N} \mid s^+_{iT} = \infty\}$. Define $\mathcal{K}^-$ in the same way. Because $s^+ \geq s^-$, we have that $K^- \subseteq K^+$. At the time horizon T, we will only change the components that are broken, so clearly

$$\sum_{i \in \mathcal{K}^+} = v_T(s^+) \geq v_T(s^-) = \sum_{i \in \mathcal{K}^-}.$$

Now for $0 \leq t \leq T-1$, the induction assumption is that $v_{t+1}(s^+) \geq v_{t+1}(s^-)$ for states such that $s^+ \geq s^-$.

We begin by showing that

$$\sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s^+, x) v_{t+1}(s') \geq \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s^-, x) v_{t+1}(s'). \tag{28}$$

Assume that $s^+_i = s^-_i, \forall i \neq l$ and $s^+_l > s^-_l$. If $x_l = 1$, the identity holds with equality, so lets assume that $x_l = 0$.

For all components except l, we create a reduced system of $N-1$ components with state space $\mathcal{S}_l$. The probability that the reduced system will end up in a state $\hat{s}$, where a set $\mathcal{J}$ of components have failed is

$$\mathbb{P}(\hat{s} \mid s, x) \quad = \quad \prod_{j \in \mathcal{J}} \mathbb{P}(j \text{ fails} \mid s_j, x_j) \prod_{j' \in \mathcal{N} \setminus \mathcal{J}} (1 - \mathbb{P}(j' \text{ fails} \mid s_{j'}, x_{j'})),$$

and we denote this probability as $\mathbb{P}(\hat{s})$. Let p^+_l and p^-_l be the probabilities that component l fails given that its age is s^+_l and s^-_l respectively. Now the expectation can be written as a sum over the states of the reduced system, i.e.,

$$\sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s^+, x) v_{t+1}(s') \quad = \quad \sum_{\hat{s} \in \mathcal{S}_l} \mathbb{P}(\hat{s}) \left(p^+_l v_{t+1}(\hat{s}^{+,\,l}) + (1 - p^+_l) v_{t+1}(\hat{s}^{+,\,\neg l}) \right)$$

where $\hat{s}^{+,\,l}$ is the state where component l has failed and the remaining components are in state $\hat{s}$. $\hat{s}^{+,\,\neg l}$ is the state where component l has not failed and the remaining components are in state $\hat{s}$. For the state s^-, the expectation is

$$\sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s^-, x) v_{t+1}(s') \quad = \quad \sum_{\hat{s} \in \mathcal{S}_l} \mathbb{P}(\hat{s}) \left(p^-_l v_{t+1}(\hat{s}^{-,\,l}) + (1 - p^-_l) v_{t+1}(\hat{s}^{-,\,\neg l}) \right)$$

We note that $\hat{s}^{+,\,l} = \hat{s}^{-,\,l}$ because if component l fails, the system will be in the same state wheter it started out in s^+ or s^-. Note also that $\hat{s}^{+,\,\neg l} \geq \hat{s}^{-,\,\neg l}$ because if component l does not fail, the age of the component will be larger if the system started out in s^+ instead of s^-. Now we want to show that

$$\Delta = \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s^+, x) v_{t+1}(s') - \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s^-, x) v_{t+1}(s') \geq 0$$

In order to simplify notation, we denote

$$\begin{aligned} v_{t+1}(\hat{s}^{+,l}) &= v_{t+1}(\hat{s}^{-,l}) = u, \\ v_{t+1}(\hat{s}^{+,\neg l}) &= u^{+}, \\ v_{t+1}(\hat{s}^{-,\neg l}) &= u^{-}. \end{aligned}$$

By using the sum over the reduced states, we get

$$\begin{aligned} \Delta &= \sum_{\hat{s}\in\mathcal{S}_l} \mathbb{P}(\hat{s}) \left(p_l^{+}u + (1-p_l^{+})u^{+}\right) \\ &- \sum_{\hat{s}\in\mathcal{S}_l} \mathbb{P}(\hat{s}) \left(p_l^{-}u + (1-p_l^{-})u^{-}\right) \\ &= \sum_{\hat{s}\in\mathcal{S}_l} \mathbb{P}(\hat{s}) \left((p_l^{+}-p_l^{-})u + (1-p_l^{+})u^{+} - (1-p_l^{-})u^{-}\right) \end{aligned}$$

Because of a nondecreasing hazard rate and $u \geq u^{+} \geq u^{-}$ from the induction assumption, we have $p_l^{+} \geq p_l^{-}$. Let $p_l^{+} = p_l^{-} + \varepsilon$, where $\varepsilon \geq 0$.

$$\begin{aligned} \Delta &= \sum_{\hat{s}\in\mathcal{S}_l} \mathbb{P}(\hat{s}) \left((p_l^{+}-p_l^{-})u + (1-p_l^{+})u^{+} - (1-p_l^{-})u^{-}\right) \\ &= \sum_{\hat{s}\in\mathcal{S}_l} \mathbb{P}(\hat{s}) \left(\varepsilon u + (1-p_l^{-}-\varepsilon)u^{+} - (1-p_l^{-})u^{-}\right) \\ &= \sum_{\hat{s}\in\mathcal{S}_l} \mathbb{P}(\hat{s})(\varepsilon \underbrace{(u-u^{+})}_{\geq 0} + (1-p_l^{-}) \underbrace{(u^{+}-u^{-})}_{\geq 0}) \\ &\geq 0, \end{aligned}$$

which proves that (28) holds. By using inequality (28), we get

$$\begin{aligned} v_t(s^{+}) &= \min_{x\in\mathcal{X}_t} \left(c(x) + \sum_{s'\in\mathcal{S}} \mathbb{P}(s' \mid s^{+}, x) v_{t+1}(s') \right) \\ &= c(x^{+}) + \sum_{s'\in\mathcal{S}} \mathbb{P}(s' \mid s^{+}, x^{+}) v_{t+1}(s') \\ &\geq c(x^{+}) + \sum_{s'\in\mathcal{S}} \mathbb{P}(s' \mid s^{-}, x^{+}) v_{t+1}(s') \\ &\geq \min_{x\in\mathcal{X}_t} \left(c(x) + \sum_{s'\in\mathcal{S}} \mathbb{P}(s' \mid s^{-}, x) v_{t+1}(s') \right) \\ &= v_t(s^{-}) \end{aligned}$$

which proves that $v_t(s^{+}) \geq v_t(s^{-})$. For states where $s_i^{+} > s_i^{-}$ for more than one component, we create a serie of inequalities $s^{+} \geq s^{1} \geq s^{2} \cdots \geq s^{-}$, where each state only differs on one component.

By the induction principle, this proves that $v_t(s)$ is nondecreasing in s for $t = 1, \ldots, T$. □

4.6 Pre- and Post-decision State Variables

As we explained section 2.2, some transition functions can be divided into two functions. One function that captures the impact of a decision, and one function that captures the impact of the exogenous information, i.e.,

$$\begin{aligned} s_t^x &= \phi^x(s_t, x_t), \\ s_{t+1} &= \phi^w(s_t^x, w_t). \end{aligned}$$

Here s_t is the state the system is in just before we make a decision, and s_t^x is the state just after we have made decision x_t. The state variable s_t is referred to as the pre-decision state variable and the state variable s_t^x is referred to as the post-decision state variable.

We assume that the transition for a component $i \in \mathcal{N}$ only depends on s_{ti} and x_{ti}. The transition function for each component is

$$\phi_i^x(s_{ti}, x_{ti}) = \begin{cases} 0, & \text{if } x_{ti} = 1, \\ s_{ti}, & \text{otherwise,} \end{cases}$$

which means that if we replace a component, we set that components age to 0. The exogenous information transition function for each component is

$$\phi_i^w(s_{ti}^x, w_{ti}) = \begin{cases} \infty, & \text{if } w_{ti} = 1, \\ s_{ti}^x + 1, & \text{otherwise.} \end{cases}$$

When using post-decision variables, the Bellman equations that we need to solve is

$$v_t^x(s_t^x) = \mathbb{E}\left(\min_{x \in \mathcal{X}_{s_t}} \left(c(x) + v_{t+1}^x(s_t^x)\right) \mid s_t^x\right) \quad \forall s_t^x \in \mathcal{S}^x, \quad \text{for } t = T, T-1, \ldots, 0$$

where $v_t^x(s_t^x)$ is the minimal expected maintenance cost for the time period $t, t+1, \ldots, T$, given that the system is in post-decision state s_t^x at time t.

4.7 Numerical Example

We consider at a system with two components. The cost of a maintenance occasion is d and the replacement costs of the components are c_1 and c_2. As explained in the model, we denote $p_i(s) = \mathbb{P}(\tau_i \leq s+1|\tau_i > s)$ as the probability that component i fails in the next time step, given that the component has age s. For our numerical example, we have that

$$\begin{aligned} p_1(0) = 0, \quad p_1(1) = 0.5, \quad p_1(2) = 1, \\ p_2(0) = 0, \quad p_2(1) = 0, \quad p_2(2) = 1, \end{aligned}$$

which means that component 2 is a deterministic component that will fail at the next time step if the age of the component is equal to 2. Component 1 is a stochastic component with probability 0.5 to fail if the age is 1 and probability 1 if the age is 2. The state space $\mathcal{S}$ is then

$$\mathcal{S} = \left\{ \underbrace{\begin{pmatrix}1\\1\end{pmatrix}}_{s^1}, \underbrace{\begin{pmatrix}1\\2\end{pmatrix}}_{s^2}, \underbrace{\begin{pmatrix}1\\\infty\end{pmatrix}}_{s^3}, \underbrace{\begin{pmatrix}2\\1\end{pmatrix}}_{s^4}, \underbrace{\begin{pmatrix}2\\2\end{pmatrix}}_{s^5}, \underbrace{\begin{pmatrix}2\\\infty\end{pmatrix}}_{s^6}, \underbrace{\begin{pmatrix}\infty\\1\end{pmatrix}}_{s^7}, \underbrace{\begin{pmatrix}\infty\\2\end{pmatrix}}_{s^8}, \underbrace{\begin{pmatrix}\infty\\\infty\end{pmatrix}}_{s^9} \right\}.$$

Because we have a system with two components, the decision space $\mathcal{X}$ is

$$\mathcal{X} = \left\{ \underbrace{\begin{pmatrix}0\\0\end{pmatrix}}_{x^1}, \underbrace{\begin{pmatrix}0\\1\end{pmatrix}}_{x^2}, \underbrace{\begin{pmatrix}1\\0\end{pmatrix}}_{x^3}, \underbrace{\begin{pmatrix}1\\1\end{pmatrix}}_{x^4} \right\}.$$

For each state s, a decision space $\mathcal{X}_s \subseteq \mathcal{X}$ can be found. For a state where no component has failed, we will not perform any maintenance and the only decision available is to choose decision x^1. In the table below, the decision spaces for all states are presented.

	s^1	s^2	s^3	s^4	s^5	s^6	s^7	s^8	s^9
$\mathcal{X}_s$	x^1	x^1	x^2, x^4	x^1	x^1	x^2, x^4	x^3, x^4	x^3, x^4	x^4

For this numerical example, we let the time horizon be $T = 2$. For simplicity, we assume that the only maintenance we will do at the time horizon is the necessary maintenance, i.e. we replace the broken components and nothing else. This is equivalent to setting $v_3(s) = 0$, for all $s \in \mathcal{S}$.

Our problem is to obtain an optimal decision, given the current system state. Assume that the initial state is $s_0 = s^3 = (1, \infty)^T$, which means that component 2 has failed and component 1 has age 1. The optimization task is to determine if we should replace only component 2 or if we should replace both components. Mathematically, this is

$$x^* = \arg \min_{x_0 \in \mathcal{X}_{s^3}} \mathbb{E} \left(\sum_{t=0}^{3} c(x_t) \,|\, s_0 = s^3 \right),$$

where $c(x) = 0$ if $x = 0$ and $c(x) = d + c^T x$ otherwise.

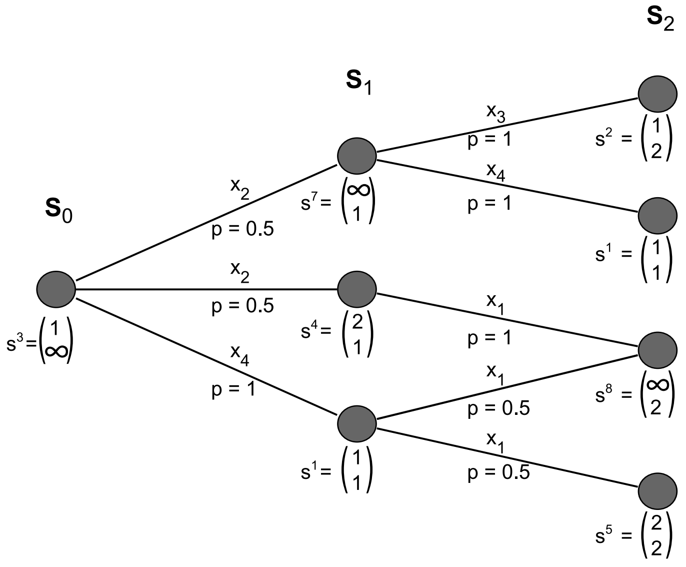
4.7.1 Backward Dynamic Programming

As we have showed earlier, this problem can be solved by solving the Bellman equations, which are

$$v_t(s) = \min_{x \in \mathcal{X}_s} \left(c(x) + \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \,|\, s, x) v_{t+1}(s') \right), \quad \forall s \in \mathcal{S} \quad \text{and} \quad t = 2, 1, 0$$

We do not need to solve Bellman's equations for all states, only for the states that the system might reach. In Figure 4, a state tree for time steps $t = 0, 1, 2$ is shown.

We begin by finding $v_2(s)$ for states



S_0
S_1
S_2
$s^3 = \begin{pmatrix}1\\ \infty\end{pmatrix}$
x_2
$p = 0.5$
x_2
$p = 0.5$
x_4
$p = 1$
$s^7 = \begin{pmatrix}\infty\\ 1\end{pmatrix}$
$s^4 = \begin{pmatrix}2\\ 1\end{pmatrix}$
$s^1 = \begin{pmatrix}1\\ 1\end{pmatrix}$
x_3
$p = 1$
x_4
$p = 1$
x_1
$p = 1$
x_1
$p = 0.5$
x_1
$p = 0.5$
$s^2 = \begin{pmatrix}1\\ 2\end{pmatrix}$
$s^1 = \begin{pmatrix}1\\ 1\end{pmatrix}$
$s^8 = \begin{pmatrix}\infty\\ 2\end{pmatrix}$
$s^5 = \begin{pmatrix}2\\ 2\end{pmatrix}$

Figure 4: State tree.

$$s^1 = \begin{pmatrix}1\\1\end{pmatrix}, \quad s^2 = \begin{pmatrix}1\\2\end{pmatrix}, \quad s^5 = \begin{pmatrix}2\\2\end{pmatrix}, \quad s^8 = \begin{pmatrix}\infty\\2\end{pmatrix}$$

The only contribution to $v_2(s)$ will be the necessary maintenance done at time $t = 2$, i.e.,

$$\begin{aligned}
v_2(s^1) &= 0,\\
v_2(s^2) &= 0,\\
v_2(s^5) &= 0,\\
v_2(s^8) &= d + c_1.
\end{aligned}$$

Now we move to time $t = 1$ and solve $v_1(s)$ for states

$$s^1 = \begin{pmatrix}1\\1\end{pmatrix}, \quad s^4 = \begin{pmatrix}2\\1\end{pmatrix}, \quad s^7 = \begin{pmatrix}\infty\\1\end{pmatrix}.$$

$$\begin{aligned}
v_1(s^1) &= 0 + 0.5v_2(s^8) + 0.5v_2(s^5)\\
&= 0.5(d + c_1),\\
v_1(s^4) &= 0 + v_2(s^5)\\
&= d + c_1,\\
v_1(s^7) &= \min_{x^3, x^4}\left(c(s,x) + \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s^7, x) v_2(s')\right)\\
&= d + c_1 + v_2(s^2)\\
&= d + c_1
\end{aligned}$$

Now we are down to time $t = 0$ and can find our optimal decision by

$$v_0(s^3) = \min_{x^2, x^4} \left(c(s,x) + \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s^3, x) v_1(s') \right)$$

The optimal decision depends on the costs d, c_1, c_2, so we look at both of our decisions.

$$\begin{aligned} v_0(s^3, x^2) &= d + c_2 + 0.5v_1(s^7) + 0.5v_1(s^4) \\ &= 2d + c_1 + c_2 \\ v_0(s^3, x^4) &= d + c_1 + c_2 + v_1(s^1) \\ &= 1.5d + 1.5c_1 + c_2 \end{aligned}$$

We see that if $d > c_1$, we would choose decision x^4 and replace both components. If $d < c_1$, we would only replace component 2. If $d = c_1$, both decisions yield the same expected maintenance cost.

4.7.2 Using Post-decision States

If we instead use post-decision state variables, we have to construct a state space for the post-decision variables. This state space is

$$\mathcal{S}^x = \left\{ \underbrace{\begin{pmatrix}0\\0\end{pmatrix}}_{s^{x,1}}, \underbrace{\begin{pmatrix}0\\1\end{pmatrix}}_{s^{x,2}}, \underbrace{\begin{pmatrix}0\\2\end{pmatrix}}_{s^{x,3}}, \underbrace{\begin{pmatrix}1\\0\end{pmatrix}}_{s^{x,4}}, \underbrace{\begin{pmatrix}1\\1\end{pmatrix}}_{s^{x,5}}, \underbrace{\begin{pmatrix}1\\2\end{pmatrix}}_{s^{x,6}}, \underbrace{\begin{pmatrix}2\\0\end{pmatrix}}_{s^{x,7}}, \underbrace{\begin{pmatrix}2\\1\end{pmatrix}}_{s^{x,8}}, \underbrace{\begin{pmatrix}2\\2\end{pmatrix}}_{s^{x,9}} \right\}$$

and a state tree for our problem is illustrated in Figure 5.

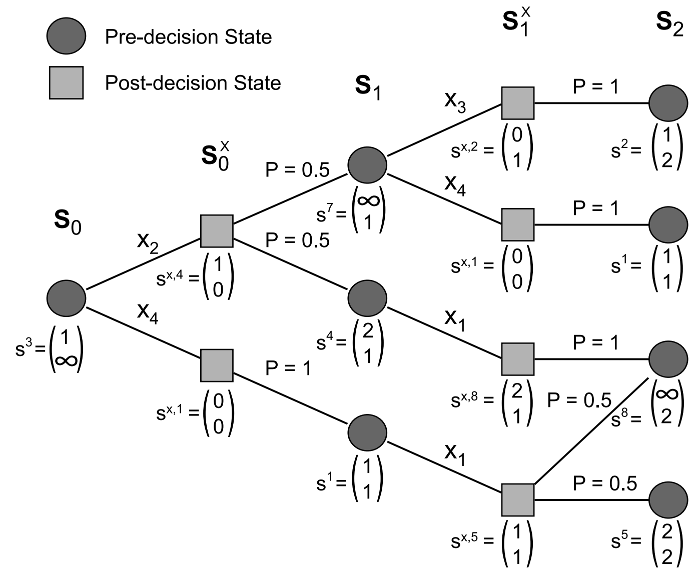

Pre-decision State
Post-decision State
S_0
S_0^x
S_1
S_1^x
S_2
x_2
x_4
x_3
x_4
x_1
x_1
P = 0.5
P = 0.5
P = 1
P = 1
P = 1
P = 1
P = 0.5
P = 0.5
$s^3 = \begin{pmatrix}1\\\infty\end{pmatrix}$
$s^{x,4} = \begin{pmatrix}1\\0\end{pmatrix}$
$s^{x,1} = \begin{pmatrix}0\\0\end{pmatrix}$
$s^7 = \begin{pmatrix}\infty\\1\end{pmatrix}$
$s^4 = \begin{pmatrix}2\\1\end{pmatrix}$
$s^1 = \begin{pmatrix}1\\1\end{pmatrix}$
$s^{x,2} = \begin{pmatrix}0\\1\end{pmatrix}$
$s^{x,1} = \begin{pmatrix}0\\0\end{pmatrix}$
$s^{x,8} = \begin{pmatrix}2\\1\end{pmatrix}$
$s^{x,5} = \begin{pmatrix}1\\1\end{pmatrix}$
$s^2 = \begin{pmatrix}1\\2\end{pmatrix}$
$s^1 = \begin{pmatrix}1\\1\end{pmatrix}$
$s^8 = \begin{pmatrix}\infty\\2\end{pmatrix}$
$s^5 = \begin{pmatrix}2\\2\end{pmatrix}$

Figure 5: State tree using post-decision states.

The backward dynamic program for post-decision states is essentially the same as for pre-decision states, but a major difference is that when solving the optimization problem at each pre-decision state, the problem is deterministic. As explained earlier, at each pre-decision state we have to solve

$$v_t(s) = \min_{x \in \mathcal{X}_s} \left(c(s,x) + v_t^x(s^x) \right)$$

which is a deterministic problem. The stochastic part of the model is captured in the value function v_t^x. When we solve small problems exactly, using post-decision state variables does not simplify the calculations, but when we later use approximative methods, the computational difference is substantial.

At time $t = 2$, we have the same values for $v_2(s)$ as those we used in section 4.7.1, i.e. $v_2(s^1) = v_2(s^2) = v_2(s^5) = 0$ and $v_2(s^8) = d + c_1$. Now we can evaluate $v_t^x(s^x)$ according to

$$v_t^x(s^x) = \mathbb{E}(v_{t+1}(s_{t+1})),$$

which gives us

$$\begin{aligned} v_1^x(s^{x,1}) &= 0, \\ v_1^x(s^{x,2}) &= 0, \\ v_1^x(s^{x,5}) &= 0.5(d + c_1), \\ v_1^x(s^{x,8}) &= d + c_1. \end{aligned}$$

We proceed to the pre-decision state at time $t = 1$ and evaluate v_1 after which we evaluate v_0^x.

$$\begin{aligned} v_1(s^1) &= 0 + v_1^x(s^{x,5}) & \qquad v_0^x(s^{x,1}) &= 0.5 v_1(s^7) + 0.5 v_1(s^4) \\ &= 0.5(d + c_1), & &= d + c_1, \\ v_1(s^4) &= 0 + v_1^x(s^{x,8}) & v_0^x(s^{x,2}) &= v_1(s^1) \\ &= d + c_1, & &= 0.5(d + c_1), \\ v_1(s^7) &= d + c_1 + v_1^x(s^{x,2}) & & \\ &= d + c_1. & & \end{aligned}$$

Now we are down to time $t = 0$, and the problem can be formulated as

$$v_0(s^3) = \min_{x^2, x^4} \left(c(s,c) + v_0^x(s^x(x)) \right),$$

which gives us the same answer as in section 4.7.1. If $d \geq c_1$ we replace both components and if $d \leq c_1$ we only replace component 2.

4.7.3 Infinite Horizon, Linear Programming

When solving an infinite horizon problem, we obtain a stationary solution. A stationary solution gives us optimal decisions for each state and is not dependent on time. As we have proven in section 2.4, the stationary problem can be reformulated as a LP. We obtain the costs $v(s)$ for each $s \in \mathcal{S}$ by solving

$$\begin{aligned} \max \quad & \beta^T v, \\ s.t. \quad & c(x) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s, x) v(s') \geq v(s), \quad \forall s \in \mathcal{S}, x \in \mathcal{X}_s, \end{aligned}$$

which is a linear program with a constraint for each state-decision pair. The problem defined in section 4.7 has 9 different states and 13 different state-decision pairs. The constraints are

$$
\begin{array}{ccccccccccc}
v_1 & & & & -0.5\gamma v_5 & & & -0.5\gamma v_8 & & \leq & 0, \\
& v_2 & & & & -0.5\gamma v_6 & & & -0.5\gamma v_9 & \leq & 0, \\
& & v_3 & -0.5\gamma v_4 & & & -0.5\gamma v_7 & & & \leq & d + c_2, \\
-\gamma v_1 & & & v_3 & & & & & & \leq & d + c_1 + c_2, \\
& & & v_4 & & & & -\gamma v_8 & & \leq & 0, \\
& & & & v_5 & & & & -\gamma v_9 & \leq & 0, \\
& & & & & v_6 & -\gamma v_7 & & & \leq & d + c_2, \\
-\gamma v_1 & & & & & v_6 & & & & \leq & d + c_1 + c_2, \\
& -\gamma v_2 & & & & & v_7 & & & \leq & d + c_1, \\
-\gamma v_1 & & & & & & v_7 & & & \leq & d + c_1 + c_2, \\
& & -\gamma v_3 & & & & & v_8 & & \leq & d + c_1, \\
-\gamma v_1 & & & & & & & v_8 & & \leq & d + c_1 + c_2, \\
-\gamma v_1 & & & & & & & & v_9 & \leq & d + c_1 + c_2.
\end{array}
$$

By solving the above linear program with $\gamma = 0.99$, and $d = 10, c_1 = 20$, and $c_2 = 10$, the following result is obtained.

$$
\begin{aligned}
v(s^1) &= 1588.8, & v(s^4) &= 1596.7, & v(s^7) &= 1610.8, \\
v(s^2) &= 1596.7, & v(s^5) &= 1596.7, & v(s^8) &= 1612.9, \\
v(s^3) &= 1607.7, & v(s^6) &= 1612.9, & v(s^9) &= 1612.9.
\end{aligned}
$$

When solving 29 with the given constraints, we do not just obtain the values v. Consider the constraints corresponding to one specific $s \in \mathcal{S}$. Each decision $x \in \mathcal{X}_s$ yields one constraint, however, only the constraint corresponding to the optimal decision is active. Considering a non-optimal decision will yield a constraint with slack. So by analyzing the dual variables of the LP, one can find the optimal policy by choosing the non-zero dual variables. By doing this, we see that the optimal decision for our initial state $s^3 = (1, \infty)^T$ is to replace only component 2.

If we, however, change the costs to $d = 30$, $c_1 = 20$ and $c_2 = 10$, which means that the service cost is more expensive relative to the component costs, we obtain a solution in which it will be optimal to replace both components.

5 Algorithms

As we have in sections 2 and 3, a number of different methods can be used to solve our problem. To measure the performance of the different algorithms we use, we will analyze how they perform against previously developed policies. The policies we will use in the comparison are a non-opportunistic policy, a policy that uses age limits to decide which components to replace, and a policy which solves a deterministic replacement problem where the deterministic lives of the components are taken to be the expected value of their real life length.

Definition 4 (non-opportunistic maintenance policy) *Replace failed components only.*

Definition 5 (age policy) *Given age limits $\hat{s}_i$ for all components $i \in \mathcal{N}$, a component $i \in \mathcal{N}$ is replaced if its age $s_i \geq \hat{s}_i$.*

When utilizing the age policy, age limits $\hat{s}_i$ for all $i \in \mathcal{N}$ have to be chosen. The heuristic algorithm for setting the age limits are explained in [1].

Definition 6 (deterministic policy) *Solve the opportunistic replacement problem with T_i being the expected value of the life of component $i \in \mathcal{N}$. Replace components according to the optimal solution at time 0.*

The deterministic policy solves a simplified deterministic problem that assumes that the components have deterministic lives equal to the expected lives of the stochastic components. By solving the deterministic replacement problem, a maintenance schedule is obtained but the only output of the policy is the decision one should take at this timestep, which corresponds to the decision at time 0.

5.1 Dynamic Programming

Given an initial state s_0, we construct a tree which denotes which states that are possible to reach in each time step. When the state-tree is constructed, we utlize dynamic programming to find the value $v_t(s_t)$ for all states s_t in the tree for all $t = T, \dots, 1$. When these values are calculated, an optimal policy is deduced. This algorithm is presented more thoroughly in section 2.3.

Definition 7 (dynamic policy) *Given a state s, backward dynamic programming (see Algorithm 2.1) is used to decide which components that should be replaced*

Even for small instances, the state tree becomes extremely large and the algorithm becomes computationally intractable.

When assuming an infinite horizon, we evaluate the value function $v(s)$ for all states $s \in \mathcal{S}$. To evaluate the value function, several methods are possible but we found that using the linear programming approach described in section 2.4.3 was most efficient.

Definition 8 (infinite horizon policy) *Evaluate the value function by using the method in section 2.4.3 for all states and use the values to decide which components to replace.*

Of course assuming an infinite time horizon is a simplifcation of the original problem, and the result is a stationary policy which does not take into account the time left to the time horizon.

5.2 Approximate Dynamic Programming

Because of the curses of dimensionality (see section 2), exact dynamic programming can not be used for larger problem. We have to make some approximations in our algorithm and one way to do so is to utilize approximate dynamic programming (see section 3). The core of approximate dynamic programming is approximating the cost function v_t by some function $\hat{V}_t$. This function can in principle take on any form but in order to update the approximation iteratively, we choose a function on the form

$$\hat{V}_t(s) = \sum_{k \in \mathcal{K}} \phi_k(s)\theta_k,$$

where ϕ_k are some basis functions that are choosen by analyzing the properties of our problem. By using a set of basis functions, the problem is reduced from evaluating the cost function $v(s)$ for all states $s \in \mathcal{S}$ to estimating a set $\mathcal{K}$ of parameters.

Linear Value Function

A first set of basis functions are $\phi_k = s_k$ for $k = 1, \ldots, N$ and $\phi_{N+1} = T - t$. This is an approximation of the cost function by a linear combination of the ages of the component and a term which takes into account the time left to the time horizon. The cost function v_t is non-decreasing in s, and our value approximation has the same property as long as $\theta_k \geq 0$ for $k = 1, \ldots, N$. This approximation gives us $N + 1$ parameters to estimate. We construct a policy that utilizes this approximation.

Definition 9 (ADP linear policy) *Utilize Algorithm 3.1 with the linear function approximation.*

Quadratic Value function

To extend the linear cost function approximation, we add a new set of basis functions that includes all possible multiples of ages $s_i s_j$ for all $i, j \in \mathcal{N}$ and a quadratic term $(T - t)^2$. This adds $N + \binom{N}{2} + 1$ new basis functions, and gives us a cost function approximation on the form

$$\hat{v}_t(s) = \sum_{i \in \mathcal{N}} \theta_i s_i + \sum_{i,j \in \mathcal{N}} \theta_{ij} s_i s_j + \theta_{t1}(T - t) + \theta_{t2}(T - t)^2, \quad (29)$$

with totally $\frac{N(N+3)}{2} + 2$ parameters to estimate. Also in this case, if the parameters θ are non-negative, the cost function approximation is non-decreasing in s.

Definition 10 (ADP quadratic policy) *Utilize Algorithm 3.1 with the quadratic function approximation.*

Approximate Version, Infinite Horizon

To simplify even further, we can use the same value function approximation as above and assume an infinite time horizon. By solving the LP described in section 3.2, we can find a vector $\tilde{r}$ such that $\Phi\tilde{r}$ is a close approximation to v.

Definition 11 (ADP infinite policy) *Estimate the parameters $\tilde{r}$ in the approximate LP. By using the approximation, find which components to replace.*

6 Results

To evaluate the performance of the different policies defined in section 5, we analyze their performance on different test problems. The evaluation is based on four different aspects;

- *Expected maintenance cost*: To minimize the expected maintenance cost is the main objective, so the evaluation focuses mainly on how the different policies perform in minimizing costs compared to eachother.
- *Standard deviation*: Because we are analyzing stochastic systems, we have to include an analysis regarding the standard deviation of the maintenance costs. A policy which has a smaller expected maintenance cost will not always be preferred if the policy has a large standard deviation.
- *Coputation time*: The computation time the policies need for producing a decision is a very important property. For example, the backward dynamic programming policy might give the best possible decision, but the policy becomes computationally intractable for larger problems.
- *Problem property*: Depending on which properties the problem instance has, different algorithms may be preferred. For example, we might expect the deterministic algorithm to perform very well in problems where the components have lives with a small standard deviation.

The evaluation is performed on four different problem instances: two test problems (T1 and T2) and the wind turbine problem (WT).

6.1 Test Problems

To examine the performance of the policies in section 5 more thouroghly, we have created two test problems (T1 and T2) that have different properties. The replacement costs and Weibull parameters for the two problems are presented in Table 1.

Table 1: The data for the test problems T1 and T2, where c_i is the replacement cost, and α_i and β_i are the Weibull parameters for component $i \in \mathcal{N}$. For T2, two of the components have deterministic lives represented by T_i.

(a) T1, $T = 30$

no.	1	2	3
c_i	2	4	6
α_i	5	7	9
β_i	6	6	6

(b) T2, $T = 30$

no.	1	2	3	4	5
c_i	2	4	6	5	8
α_i	5	7	9	-	-
β_i	6	6	6	-	-
T_i	-	-	-	6	8

The problem we will focus mainly on is T1 because the size of the problem allows us to utilize and compare the performance of all different policies. In Figure 6, the total mean maintenance cost for T1 are presented for different maintenance occasion costs d. The results are weighted against the maintenance cost for the non-opportunistic maintenance policy and clearly, when the maintenance occasion cost increases, the reduction in maintenance cost for the opportunistic policies increases. For problem T1, the exact dynamic programming algorithm gives superior results for all different maintenance costs. When the maintenance cost d is more than three

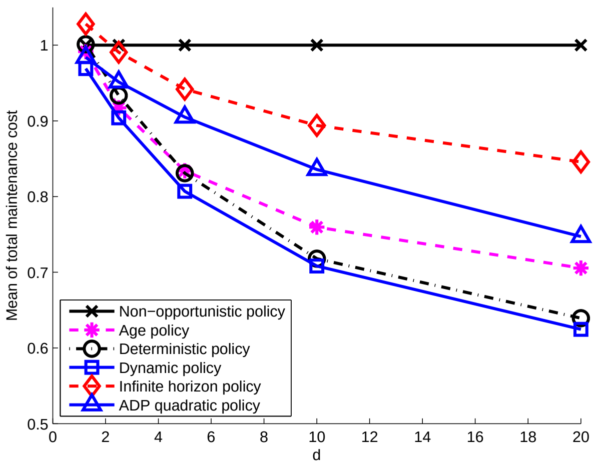


Figure 6: Mean maintenance cost for problem T1 for six different algorithms where the maintenance occasion costs d is varied.

times the cost for the most expensive component, the dynamic programming policy reduces the maintenance cost with almost 40%. The infinite horizon policy, which obtains a stationary policy, gives higher maintenance costs than the non-opportunistic policy when the maintenance occasion costs are small compared to the component costs.

In Figure 7, the mean total maintenance costs for the different policies are presented for the problem T1 where the Weibull β-parameter is varied. We note that the exact dynamic programming algorithm still produces a maintenance policy with minimial maintenance cost for all different scenarios. Note that when $\beta = 1$ for all components, the optimal policy will actually be a non-opportunistic policy. The reason for this is that the hazard rate (failure rate) for the components are constant, which means that the age of the components does not affect the maintenance decision. The optimal decision will always be to only replace the necessary components. When β increases, the components lives becomes less stochastic and when $\beta = 8$ for all components, the deterministic maintenance policy produces the same results as the exact dynamic programming policy. The computation time for the algorithms for T1 can be seen in Table 2.

Table 2: The computation time in seconds for the policies to compute a decision.

	Age	Deterministic	Dynamic	Infinite	ADP
$T = 30$	< 1	< 1	≈ 300	≈ 45	≈ 60
$T = 10$	< 1	< 1	≈ 20	≈ 45	≈ 10

In Figure 8, the total mean maintenance costs for T2 are presented for different maintenance occasion costs d. When adding two new components, the size of the problem becomes too large for exact dynamic programming to be utilizable, so the evaluation is performed on only four different maintenance policies; the non-opportunistic policy, the deterministic policy, the age policy and the ADP policy. We note that the ADP policy produces lower maintenance costs than the non-opportunistic policy. However, both the deterministic policy and the age policy gives superior results compared with the ADP policy.

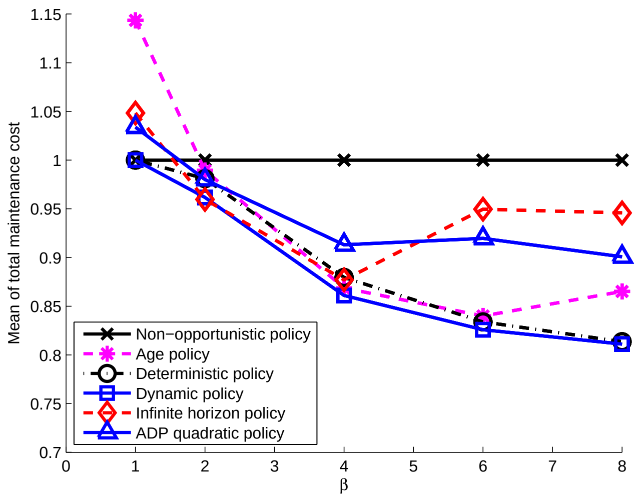


Figure 7: Mean maintenance cost for problem T1 for six different algorithms where the maintenance occasion costs β is varied.

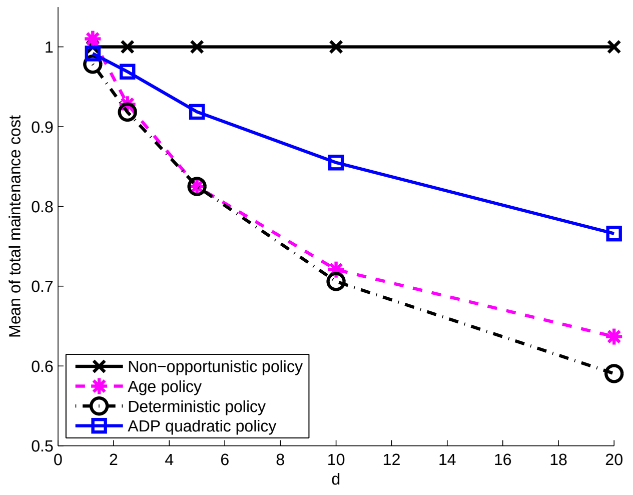


Figure 8: Mean maintenance cost for problem T2 evaluated for different maintenance occasion costs d.

6.2 Wind Turbine Problem (WT)

The wind turbine problem is based on a study made by Poore and Walford [11], and concerns a land based wind turbine unit. The maintenance considered requires a construction crane to be used, which makes the maintenance occasion cost $d = \$30000$, a relatively high cost compared to the component costs. 14 components are considered, and their lives are all modelled by the Weibull distribution. In Table 3, the components replacement cost and Weibull parameters are presented. As can be seen in the table, many of the components have shape-parameter $\beta = 1$, which means that they have a constant failure rate. The reason for this is that the failures of these components depend on external factors and not on the component age. If we have a system with only components with these property, the optimal policy would be to only replace the failed components at each maintenance occasion (i.e. non-opportunistic maintenance). However, there exists components with non-decreasing failure rates in the system, so the problem instance is interesting from our point of view. Because some of the components have a constant failure rate, we will only replace those components when they fail. This means that we can reduce the decision space at each maintenance occasion to a problem where we need to make a decision regarding only four components. In figure 9, the mean maintenance costs for the WT is presented.

Table 3: The problem parameters for the wind turbine (WT), where c_i is the replacement cost and α_i and β_i are the Weibull parameters for component i

no.	1-3	4	5-7	8	9	10	11	12	13-14
c_i	101	48	43	91	122	85	137	96	36
α_i	400	20	400	400	400	20	20	400	17
β_i	1.0	1.0	1.0	1.0	1.0	3.5	3.5	1.0	3.5

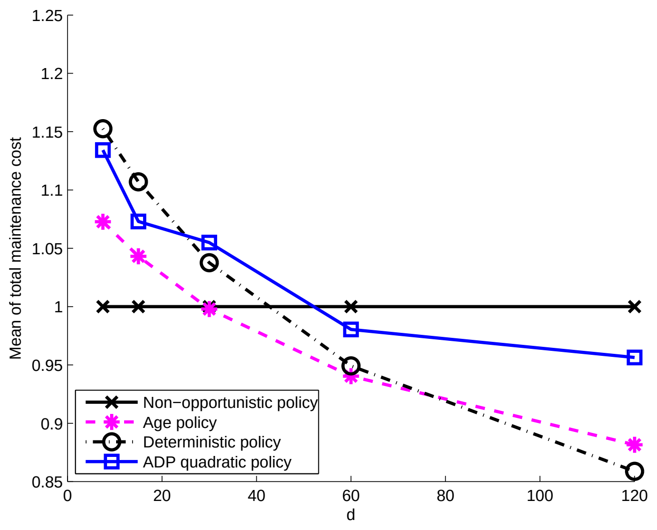


Figure 9: Mean maintenance cost for the Wind Turbine problem evaluated for different maintenance occasion costs d.

7 Conclusions and Future Research

The policy based on an exact dynamic programming algorithm gives superior results regarding maintenance cost compared to other maintenance policies. The drawback with this algorithm is the computation time. Even for medium size problems, an exact dynamic programming algorithm will be computationally intractable. One method for decreasing the computational time for a backward dynamic programming algorithm is to construct a ”state tree” which includes all possible states the system might visit in the future. By constructing state trees, the compuation time was reduced with up to 40%. Even with such an implementation, the algorithm can not manage to compute a decision within 5 minutes for problem instances with 5 or more components and time horizons over 20 time steps.

To overcome the computational difficulties, approximate dynamic programming (ADP) has been analyzed. The main challenge when utilizaing ADP is to find good approximations to the value function. We have used two different approximations in this thesis; a linear function and a quadratic function. The results of the linear function were not satisfactory and did not produce any opportunistic maintenance decisions. The quadratic form gave better results, but still not comparable to the deterministic policy. Other approximations might give better maintenance policies, and I believe that a function on the form

$$\hat{V}(s) = \sum_{k=1}^{K} \theta_{k1} s_k^{\theta_{k2}}$$

will give better approximations for the value function. The reason for not evaluating such an approximation is that to update the parameters θ_{1k} and θ_{2k}, updating techniques based on solving linear systems can not be utilized. For such value approximations, other techniques such as neural network methods have to be utilized. Not only the form of the approximating function is important in ADP. How to update the function is not always obvious. Most common updating techniques is to use a smoothing operator which updates the cost function values according to

$$\hat{V}^n = (1 - \alpha_{n-1})\hat{V}^{n-1} + \alpha_{n-1}\hat{v}^n,$$

where $\hat{v}^n$ is the cost observation at iteration n and α_n is some suitable step length. Such an updating rule is easy to implement and often gives satisfactory results. One problem is that the realization of $\hat{v}^n$ depends on the values $\hat{V}^{n-1}$, which means that $\hat{v}^n$ is a biased observation. To remove such biases, one can use an updating rule which puts more weight on later observations.

Another way of reducing the problem is to assume an infinite horizon. Even though this is a large simplification, the method should be able to produce near optimal maintenance policies for problems where the planning period is long relative to the components lives. But, on the other hand, for shorter planning periods, the method will probably do more maintenance than what is optimal. Therefore, a combination between an infinite and a finite time horizon method, in which the infinite method is applied when the time to the horizon is long and the finite horizon method is utilized when the time left is short, could prove efficient.

One aspect that we have not analyzed in this thesis is the possibility to do better implementations of the algorithms. All algorithms used have been implemented in MATLAB and almost no effort has been concentrated on reducing computational times. For example, the backward dynamic programming model can with our implementation only handle very small problems with planning periods up to 50-100 time steps and systems with 3-5 components. With a better implementation, the algorithm might be able to handle larger problem instances (5-10 components). It should be pointed out that the focus of the thesis was not to find applicable mathods for real life problems, but rather to investigate the possibility of utilizing a dynamic programming model for the SORP.

References

[1] T. Almgren, N. Andréasson, M. Patriksson, A-B. Stromberg, A. Wojciechowski, *The replacement problem: A Poluhedral and complexity analysis. The complete version*, Preprint - Department of Mathematical Sciences, Chalmers University og Technology and Gothenburg University, Gothenburg, 2009

[2] N. Andréasson, A. Evgrafov, and M. Patriksson, *An Introduction to Continuous Optimization*, Studentlitteratur, Lund, 2005.

[3] R. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

[4] B. Dickman, S. Epstein, Y. Wilamowsky, *A mixed integer linear programming formulation for multi-component deterministic opportunistic replacement*, The Journal of the Operational Research Society of India, 28 (1991), pp. 165-175.

[5] D. P. de Farias, A. van Roy, *The Linear Programming Approach to Approximate Dynamic Programming*, Operations Research, Vol. 5, No. 6, 2003

[6] D. P. de Farias, A. van Roy, *On Constraint Sampling in the Linear Programming Approach to Approximate Dynamic Programming*, Mathematics of Operations Research, Vol. 29, No. 3, 2004

[7] S. E. Dreyfus, A. M. Law, *The Art and Theory of Dynamic Programming*, Academic Press, Inc., New York, NY, 1977.

[8] D. W. Jorgenson, R. Radner, *Optimal Replacement and inspection of stochatically failing equipment*, Paper P-2074, Rand Corporation, Santa Monica, CA, USA, 1960.

[9] A. W. Marshall, I. Olkin, *Inequalities: Theory of Majorization and Its Applications*, Academic Press, Inc., New York, NY, 1979.

[10] M. Patriksson, A-B. Stromberg, A. Wojciechowski, *The stochastic opportunistic replacement problem: A two stage solution approach*, Chalmers University of Technology and Gothenburg University, Gothenburg, 2010.

[11] W. B. Powell, *Development of an operations and maintenance cost model to identify cost of energy savings for low wind speed turbines*, Subcontract Reposrt NREL/SR-500-40581, National Renewable Energy Laboratory, Golden, CO, USA, 2008.

[12] W. B. Powell, *Approximate Dynamic Programming*, John Wiley & Sons, Inc., Hoboken, NJ, 2007.

[13] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic rogramming*, John Wiley & Sons, Inc., Hoboken, NJ, 1994.

A Theorems

A.1 Optimality Equations

The connections between the objective and the optimality equations in dynamic programming is essential. Even though they might seem intuitively obvious, it is very important to prove them thouroghly. We begin by creating a function

$$F_t^\pi(s_t) = \mathbb{E}\left(\sum_{t'=t}^{T-1} c_{t'}(s_{t'}, x_{t'}^\pi(s_{t'})) + c_T(s_T) \mid s_t\right).$$

If we take $t = 0$ we have the expected total cost over the whole period for a specific policy π. The main purpose of this section is to prove the relation between F_t^π and the optimality equations

$$v_t^\pi(s_t) = c_t(s_t, x_t^\pi(s_t)) + \mathbb{E}\left(v_{t+1}^\pi(s_{t+1}) \mid s_t\right).$$

We begin by proving the following theorem

Theorem A.1 $F_t^\pi(s_t) = v_t^\pi(s_t)$

Proof We will prove the theorem by induction. First we note that $F_T^\pi(s_T) = v_T^\pi(s_T) = c_T(s_T)$ for all $s_T \in \mathcal{S}$. Next we assume that the theorem holds for $t+1, t+2, \ldots, T$. We have that

$$\begin{aligned}
v_t^\pi(s_t) &= c_t(s_t, x_t^\pi(s_t)) + \mathbb{E}\left(v_{t+1}(s_{t+1}) \mid s_t\right)\\
&= c_t(s_t, x_t^\pi(s_t)) + \mathbb{E}\left(F_{t+1}(s_{t+1}) \mid s_t\right)\\
&= c_t(s_t, x_t^\pi(s_t)) + \mathbb{E}\left(\mathbb{E}\left[\sum_{t'=t+1}^{T-1} c_{t'}(s_{t'}, x_{t'}^\pi(s_{t'})) + c_T(s_T) \mid s_{t+1}\right] \mid s_t\right)\\
&\qquad \text{(by using the tower property)}\\
&= c_t(s_t, x_t^\pi(s_t)) + \mathbb{E}\left[\sum_{t'=t+1}^{T-1} c_{t'}(s_{t'}, x_{t'}^\pi(s_{t'})) + c_T(s_T) \mid s_t\right]\\
&\qquad \text{(the expectation is conditioned on } s_t\text{, so } c_t(s_t, x_t^\pi(s_t)) \text{ is deterministic)}\\
&= \mathbb{E}\left[\sum_{t'=t}^{T-1} c_{t'}(s_{t'}, x_{t'}^\pi(s_{t'})) + c_T(s_T) \mid s_t\right]\\
&= F_t^\pi(s_t)
\end{aligned}$$

□

Next we define

$$F_t^*(s_t) = \min_{\pi \in \Pi} F_t^\pi(s_t)$$

We have assumed earlier that if we solve Bellman's equations

$$v_t(s_t) = \min_{x \in \mathcal{X}}\left(c_t(s_t, x) + \sum_{s' \in \mathcal{S}} p(s' \mid s_t, x) v_{t+1}(s')\right)$$

we will find the polixy that optimizes F_t^π. Here is the proof why such an assumption is valid.

Theorem A.2

$$F^*_t(s_t) = v_t(s_t). \tag{30}$$

Proof We will prove the theorem in two steps. First we will show that $v_t(s_t) \leq F^*_t(s_t)$ for all $s_t \in \mathcal{S}$ and $t = 0,1,\ldots,T-1$. Clearly, $v_T(s_T) = F^*_T(s_T)$ for all s_T, so we assume that $v_{t'}(s_{t'}) \leq F^*_{t'}(s_{t'})$ for $t' = t+1, t+2, \ldots, T$, and let π be any policy. We have that

$$\begin{aligned}
v_t(s_t) &= \min_{x\in\mathcal{X}} \left(c_t(s_t,x) + \sum_{s'\in\mathcal{S}} p(s' \mid s_t, x) v_{t+1}(s') \right) \\
&\leq \min_{x\in\mathcal{X}} \left(c_t(s_t,x) + \sum_{s'\in\mathcal{S}} p(s' \mid s_t, x) F^*_{t+1}(s') \right) \\
&\leq \min_{x\in\mathcal{X}} \left(c_t(s_t,x) + \sum_{s'\in\mathcal{S}} p(s' \mid s_t, x) F^\pi_{t+1}(s') \right), \quad \text{for arbitrary } \pi \\
&\leq c_t(s_t, x^\pi_t(s_t)) + \sum_{s'\in\mathcal{S}} p(s' \mid s_t, x^\pi_t(s_t)) F^\pi_{t+1}(s') \\
&= F^\pi_t(s_t), \text{by Theorem A.1.}
\end{aligned}$$

So we have that $v_t(s_t) \leq F^\pi_t(s_t)$ for any policy $\pi \in \Pi$, which proves that $v_t(s_t) \leq F^*_t(s_t)$.

Now we will prove that for any $\epsilon > 0$ there exists a policy π that satisfies

$$F^\pi_t(s_t) + (T-t)\epsilon \leq v_t(s_t).$$

We note that it is possible to find a decision x such that the following holds

$$v_t(s_t) \geq c_t(s_t,x) + \sum_{s'\in\mathcal{S}} p(s' \mid s_t, x)) v_{t+1}(s') + \epsilon. \tag{31}$$

We assume that for any $\epsilon > 0$, there exists a policy π such that $v_{t'}(s_{t'}) \leq F^\pi_{t'}(s_{t'}) + (T-t')\epsilon$ for all states s_t and for $t' = t+1, t+1, \ldots, T$. By using Theorem A.1 and the induction hypothesis we get

$$\begin{aligned}
F^\pi_t(s_t) &= c_t(s_t, x^\pi_t(s_t)) + \sum_{s'\in\mathcal{S}} p(s' \mid s_t, x^\pi_t(s_t)) F^\pi_{t+1}(s') \\
&\leq c_t(s_t, x^\pi_t(s_t)) + \sum_{s'\in\mathcal{S}} p(s' \mid s_t, x^\pi_t(s_t)) \left[v_{t+1}(s') - (T-(t+1))\epsilon \right] \\
&= c_t(s_t, x^\pi_t(s_t)) + \sum_{s'\in\mathcal{S}} p(s' \mid s_t, x^\pi_t(s_t)) v_{t+1}(s') \\
&- \sum_{s'\in\mathcal{S}} p(s' \mid s_t, x^\pi_t(s_t)) \left[(T-(t+1))\epsilon) \right] \\
&= \underbrace{\left[c_t(s_t, x^\pi_t(s_t)) + \sum_{s'\in\mathcal{S}} p(s' \mid s_t, x^\pi_t(s_t)) v_{t+1}(s') + \epsilon \right]}_{\leq v_t(s_t) \text{ by equation 31}} - (T-t)\epsilon \\
&\leq v_t(s_t) - (T-t)\epsilon.
\end{aligned}$$

So we get

$$F^*_t(s_t) + (T-t)\epsilon \leq F^\pi_t(s_t) + (T-t)\epsilon \leq v_t(s_t) \leq F^*_t(s_t),$$

which proves the theorem. □