Applying modified policy iteration to multi-component system maintenance scheduling

Petri Koivisto

School of Science

Bachelor's thesis

Espoo 11.5.2022

Supervisor

Prof. Antti Punkka

Advisor

M.Sc. Jussi Leppinen

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the bachelor's thesis

Author Petri Koivisto

Title Applying modified policy iteration to multi-component system maintenance scheduling

Degree programme Engineering Physics and Mathematics

Major Mathematics and System Sciences **Code of major** SCI3029

Teacher in charge Prof. Antti Punkka

Advisor M.Sc. Jussi Leppinen

Date 11.5.2022 **Number of pages** 25 **Language** English

Abstract

Most technical systems consist of numerous components, that require regular maintenance in order to keep the system functional. Since a single component malfunction can render a whole system inoperable, planning of the system maintenance is usually necessary in order to maintain the system in a reliable working condition. Moreover, as both maintenance operations and system downtime induce expenses, maintenance optimization can be utilized to minimize long-term system maintenance costs.

In this thesis a maintenance scheduling optimization model is examined. The model allows modeling economical and structural dependences of the system. The evolution of the system is modeled as a Markov decision process, for which an optimal maintenance policy can be calculated in order to minimize maintenance costs. This thesis implements a modified policy iteration algorithm, that can be used to calculate an optimal maintenance policy.

The performance of the modified policy iteration algorithm is compared to policy iteration algorithm by calculating optimal maintenance policies for an imaginary example system. The results indicate that the modified policy iteration algorithm outperforms the policy iteration algorithm, as over a ten-fold speed-up in computation time was achieved in certain computing cases. In addition, the modified policy iteration algorithm implementation requires only a fraction of the memory needed by the policy iteration algorithm.

Keywords Maintenance optimization, multi-component system, Markov decision process, modified policy iteration

Aalto University
School of Science

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Tekniikan kandidaatintyön tiivistelmä

Tekijä Petri Koivisto

Työn nimi Monikomponenttisen järjestelmän huollon aikatauluttaminen muokatulla ohjauksen iterointialgoritmilla

Koulutusohjelma Teknillinen fysiikka ja matematiikka

Pääaine Matematiikka ja systeemitieteet **Pääaineen koodi** SCI3029

Vastuuopettaja Prof. Antti Punkka

Työn ohjaaja DI Jussi Leppinen

Päivämäärä 11.5.2022 **Sivumäärä** 25 **Kieli** Englanti

Tiivistelmä

Tekniset järjestelmät kuluvat käytössä, minkä vuoksi niiden säännöllinen huoltaminen on välttämätöntä järjestelmän toimintakyvyn ylläpitämiseksi. Koska yksittäinen komponenttivika voi aiheuttaa koko järjestelmän toimintakyvyttömyyden, on järjestelmän huoltojen suunnittelu tärkeää. Huollon optimoinnilla voidaan minimoida ylläpitokustannukset, jotka syntyvät paitsi järjestelmän huoltotöistä myös mahdollisen vikaantumisen aiheuttamasta tuotannonmenetyksestä.

Tässä opinnäytetyössä esitellään huollon aikataulutuksen optimointimalli, joka ottaa huomioon järjestelmän rakenteelliset ja taloudelliset riippuvuudet. Järjestelmän tilan kehitystä ja huoltojen ajankohtia mallinnetaan Markov-päätösprosessilla. Markov-päätösprosessin optimaalisen ohjauksen löytämiseksi työssä implementoitiin muokattu ohjauksen iterointialgoritmi (engl. modified policy iteration).

Muokatun ohjauksen iterointialgoritmin suorituskykyä verrattiin ohjauksen iterointialgoritmiin laskemalla optimaalisia huoltoaikatauluja mallinnettavalle tekniselle järjestelmälle. Tulosten perusteella muokattu ohjauksen iterointialgoritmi suoriutuu optimiohjauksen laskemisesta tietyissä laskentatapauksissa yli kymmenen kertaa nopeammin kuin ohjauksen iterointialgoritmi. Lisäksi muokattu ohjauksen iterointialgoritmi käyttää vain murto-osan ohjauksen iterointialgoritmin vaatimasta muistimäärästä.

Avainsanat Ylläpidon optimointi, monikomponenttijärjestelmä, Markov-päätösprosessi, muokattu ohjauksen iterointialgoritmi

Contents

1 Introduction

Modern technical systems consist of multiple components, which require regular maintenance to retain the system functional. These components may have numerous operating principles, which as a result cause the components to have diverse life spans. The fact that components have different and also uncertain lifetimes makes planning of the system maintenance difficult, as a failure of a component can cause substantial expenses. The problem of system maintenance planning becomes more complex as the size of the system grows. In addition, there may exist dependences between the components, which adds another layer of complexity to the problem.

When planning a maintenance schedule for a system described above, one can intuitively find several different maintenance strategies. For example, a high reliability of the system can be achieved by replacing every component of the system at each maintenance instance, which will result in high maintenance costs. On the other hand, one can minimize short-term costs by committing to no maintenance, which will eventually lead into system failure. In the scope of this thesis, objective of maintenance optimization is to find a scheduling for system maintenances, i.e. maintenance policy, that minimizes long-term cumulative cost of system maintenance while keeping the reliability of the system above a preset reliability threshold.

This thesis presents a maintenance optimization model developed by Leppinen (2020). The model describes a system maintenance scheduling problem as a Markov decision process (e.g. Howard, 1960), for which an optimal policy can be found. In this thesis we implement a modified policy iteration algorithm, that can be used to solve the optimal maintenance policy. In addition, we compare the performance of the modified policy iteration algorithm to policy iteration algorithm, that was used by Leppinen (2020).

The remainder of this thesis is organized as follows: In Section 2 we review scientific literature in the topics of maintenance optimization and dynamic programming. In Section 3 we present the maintenance optimization model, Markov decision process and the modified policy iteration algorithm. In Section 4 we compare performances of modified policy iteration and policy iteration algorithms. In Section 5 we conclude the results of this thesis and give suggestions for future research.

2 Background

2.1 System maintenance optimization

During the last few decades, the subject of *maintenance optimization* has been under a substantial amount of research (de Jonge and Scarf, 2020; Dekker, 1996). Automatization and mechanization of production systems have reduced the number of production personnel but in contrast raised the investments to equipment and maintenance costs: for example, according to Bevilacqua and Braglia (2000) maintenance costs can represent 15–70 % of total production costs. While industrial

systems have become increasingly reliable, component wear and exposure to production environment can still cause unpredictable system failures, which as a result can lead to expenses due to production losses and potential safety issues. These aspects have lead to a wide recognition of the importance of maintenance optimization as a business function (de Jonge and Scarf, 2020).

In general, the aim of maintenance optimization is to develop and apply mathematical models for improving and optimizing maintenance policies. de Jonge and Scarf (2020) present different *optimality criteria* that can be the objectives of maintenance optimization; these objectives can be, for example, minimizing total system operation costs or maximizing system availability. The objective is achieved by selecting an optimal set of *maintenance actions*, which can be interpreted as the decision variables of the optimization problem.

There are two types of maintenance actions: *preventive* and *corrective* maintenance (e.g. Ben-Daya et al., 2016), where preventive maintenance is applied before a failure of a system or a component and corrective maintenance is applied after failure. Figure 1 illustrates the costs of corrective and preventive maintenance as different levels of preventive maintenance is applied. The figure implies that there is a minimum combined cost level of preventive maintenance. According to de Jonge et al. (2015), preventive maintenance can also significantly reduce system downtime. Preventive maintenance policies can be further divided into *time-based* and *condition-based* strategies, where the latter usually requires some type of inspections or monitoring in order to determine the condition of the system (de Jonge and Scarf, 2020).

Maintenance optimization models have been applied to both *single-component* and *multi-component* systems (de Jonge and Scarf, 2020). In multi-component systems there may exist *dependences* between the components of the system. Particularly, three types of dependence are commonly covered in maintenance optimization lit-

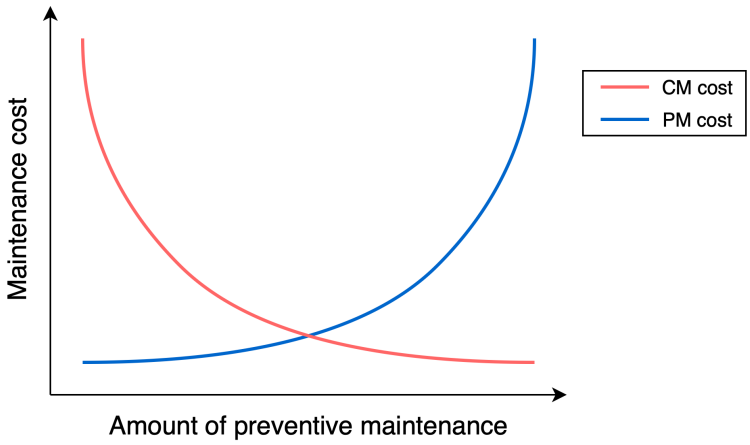

Figure 1: Illustration of corrective maintenance (CM) and preventive maintenance (PM) costs as different levels of preventive maintenance is applied (Ben-Daya et al., 2016).

erature: *economic*, *structural* and *stochastic dependence* (e.g. de Jonge and Scarf, 2020; Thomas, 1986). For example, economic dependence can arise from shared maintenance set-up costs for the components, structural dependence may result from some components having to be dismantled before maintenance of other components, and stochastic dependence may be caused by some external factor affecting the reliability of multiple components of the system. Olde Keizer et al. (2017) also mention a fourth type dependence, *resource dependence*, which can originate from a limited stock of spare parts that are shared with multiple components, or from a limited amount of repair time available. These dependences and combinations of them are incorporated in numerous maintenance optimization models (de Jonge and Scarf, 2020).

In the maintenance optimization literature, different mathematical approaches have been used to find optimal maintenance policies for multi-component systems. For example, simulation has traditionally been used in order to compare various heuristically determined maintenance policies, when the evolution of the system is uncertain (de Jonge and Scarf, 2020). However, as computation speeds have increased, more exact methods have been developed to find optimal maintenance policies. *Dynamic programming* approaches offer such tools, and those have lately been used in maintenance optimization models (e.g. Leppinen, 2020).

2.2 Dynamic programming

Dynamic programming (Bellman, 1957) can be described as a mathematical framework for solving complex mathematical optimization problems by reducing them into more easily solvable sub-problems. According to Bellman (1957), dynamic programming can be used to achieve an optimal sequence of decisions for a multi-stage decision processes. As a type of a multi-stage decision process, Bellman (1957) mentions a *Markov decision process* (MDP), which has been used to model complex decision problems in diverse fields such as ecology, economics and engineering (Puterman, 1994). In maintenance optimization literature, for example Maillart (2006) and Leppinen (2020) apply Markov decision process in the objective of minimizing long-term cumulative cost of system maintenance.

Markov decision processes are well-suited for modelling maintenance decisions for a technical system, as MDPs can be used to model the probabilistic evolution of the system. The Markov decision processes are also widely studied and are mathematically robust method (Puterman, 1994). In addition, Markov decision processes can be used to analyze decision problems with an infinite time-horizon, and many problems arising from real-life situations can be formulated in such way, that the Markov decision process always has a mathematically optimal policy that can be obtained algorithmically (Puterman, 1994).

Different algorithms have been developed for finding the optimal policy of a Markov decision process. Howard (1960) presents a *value iteration* algorithm, that calculates an approximated optimal policy of a preset accuracy. Howard (1960) also presents a

policy iteration algorithm that can be used to find the exact mathematically optimal policy for the MDP. Puterman and Shin (1978) present a *modified policy iteration* algorithm that is based on the idea of using value iteration to approximate results needed in the policy iteration algorithm iteration steps. The policy iteration and modified policy iteration algorithms are later covered in Section 3.3.

3 Methods

In this section we present a maintenance optimization model developed by Leppinen (2020). The model can be used to find a minimum cost maintenance policy for a multi-component system incorporating economic and structural dependences. Sections 3.1 and 3.2 present the model, including slight improvements to necessary probability calculations. Sections 3.3 and 3.4 are the key contributions of this thesis, where we describe the modified policy iteration algorithm and implementation of it in MATLAB.

3.1 Maintenance optimization model

In this thesis we examine a technical system that consists of n components denoted by $N_i \in N,\ i \in \{1, \ldots, n\}$. The system is considered to be a *series system*, which means that a failure of any component will cause the system to be inoperable. We assume, that the system is only maintained at discrete *maintenance instances* $t_k,\ k \in \mathbb{N}$. The difference between two maintenance instances is called the *maintenance interval* and denoted by $\Delta t = t_{k+1} - t_k > 0$. The maintenance interval can either be a fixed time interval or it can be based on the system operation time or travel distance. Moreover, we assume that during the maintenance instances the components can only be replaced with new ones: no other maintenance procedures are considered in the scope of this thesis.

Each component of the system has its own known failure probability distribution with a cumulative distribution function denoted by $F_i(t)$, which gives the probability that component N_i will fail before age t. The components of the system are assumed to be stochastically independent in the sense that they will fail independently of each other. For the system we require a *reliability threshold* ρ that gives the minimum reliability level for the system between maintenance instances. For example, if $\rho = 0.9$, at maintenance instance t_k, we require that the probability that the system is functional at maintenance instance t_{k+1} has to be at least 90 %; if the probability is lower, then maintenance is required to increase the system reliability, since the failure rate of each component is assumed to be increasing. The mathematics of the system reliability are explained in Section 3.2.

The system can have structural and economical dependences between the components. Such system and its dependences can be modeled as a *directed graph*, as presented by Leppinen (2020). In the graph each node $i \in \{0, \ldots, n\}$ describes replacement or other maintenance operation of component N_i in the system. Node 0 is called the

root node, that describes the starting point for the system maintenance. Each arc (i, j) between the nodes i and j respectively describes dependences in the system. If component N_i in the graph is accessible directly from the root node, there are no structural dependences for replacing the component. However, if component N_j is only accessible from node describing component N_i, then N_j can only be replaced if N_i is replaced also. In addition, the weights of the arcs describe the *cost* of each component replacement or other maintenance operation, denoted by c_{ij} which is the cost of maintenance operation j given that maintenance operation i is conducted before it.

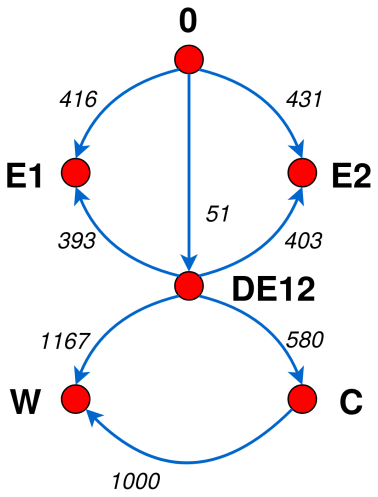

Figure 2: An example of a directed graph illustrating a system with structural and economical dependences.

An example of such graph is presented in Figure 2. The graph illustrates the structural and economical dependences of an imaginary transportation system: the nodes E1 and E2 represent two engines of the system, node DE12 represents a disassembly of both engines and the nodes W and C describe the wheels and chassis of the system, respectively. From the graph we see that the wheels or the chassis cannot be replaced without disassembling the engines first. We also see that if for example E1 is replaced without disassembling it first, the cost of the maintenance is 416 units. However, if the engines are disassembled first, the cost of engine replacement is 393 units but since the disassembly costs 51 units, the total cost of such operation is 444 units.

In addition to the costs c_{ij} the model includes a *fixed set-up cost* $c_0 \geq 0$. The set-up cost is paid at each maintenance instance if any maintenance operations are carried out. Thus it simulates for example the cost of the system being inoperable during maintenance. The model can also incorporate a *corrective surplus* cost $r_i \geq 0$ which describes the extra cost that may result from component N_i breaking between maintenance instances.

In the next section we describe modeling the evolution of the system while taking into account the effect of the maintenance operations executed at maintenance intervals.

This is achieved by modeling the state and maintenance operations as a *discrete time Markov decision process*, as developed by Leppinen (2020).

3.2 Markov decision process

Let S denote a state-space of the system and $s_k \in S$ be the state vector of the system at maintenance instance t_k. The vectors s_k can be described by two factors: the age of the components $a_k \in \mathbb{R}^n$, where $(a_k)_i$ describes the age of component N_i, and the failure state of the components $f_k \in \{0,1\}^n$, where $(f_k)_i = 1$ if component N_i has failed. Thus the state vector of the system can be written as $s_k = [a_k^\top, f_k^\top]^\top \in \mathbb{R}^{2n}$.

During each maintenance interval Δt the states of the components evolve by either ageing or failing. The *failure time* t_i^f of component N_i follows a probability distribution with cumulative distribution function $F_i((a_k)_i)$. Thus, if component N_i operates at maintenance instance t_k having age $(a_k)_i$, then it operates until t_{k+1} with the conditional probability

$$P_k^i\left(t_i^f > t_{k+1} | t_i^f > t_k\right) = \frac{1 - F_i((a_k)_i + \Delta t)}{1 - F_i((a_k)_i)} := R_i((a_k)_i). \tag{1}$$

This is *the reliability of component* N_i at maintenance instance t_k given its age $(a_k)_i$. We assume that the reliability of a component decreases as the component ages. Because in a series system every component is critical and the failures are independent, the reliability of the system at t_k given component ages a_k is

$$R_{\text{sys}}(a_k) = \prod_{i=1}^{n} R_i((a_k)_i). \tag{2}$$

Because system failure is unwanted, we set a reliability threshold $\rho \in (0,1)$ for the system. At every maintenance instance, the system must be maintained so that the system reliability in (2) remains above the threshold until the next maintenance instance meaning that

$$R_{\text{sys}}(a_k) \geq \rho \tag{3}$$

for all $k \in \mathbb{N}$.

We assume that at most one component can fail during each maintenance interval (t_k, t_{k+1}). This is reasonable since after a component fails the system becomes inoperable which prevents other component failures. The probability of no failures occurring is given by (2). Component N_i will fail during (t_k, t_{k+1}) if it is the first component that fails. We denote this as event $E_{i,k}$. Mathematically it means that failure time t_i^f of component N_i satisfies

$$t_k \leq t_i^f = \min\left\{t_1^f, \ldots, t_n^f\right\} \leq t_{k+1}.$$

In order for event $E_{i,k}$ to take place every component must operate until t_k given ages a_k. When component ages a_k and the maintenance interval Δt are known, the probability of $E_{i,k}$ can be calculated as a conditional probability

$$P(E_{i,k}(a_k)) = \frac{\int_0^{\Delta t} F_i'((a_k)_i + t) \prod_{j \neq i} [1 - F_j((a_k)_j + t)] \, \mathrm{d}t}{\prod_{i=1}^{n} [1 - F_i((a_k)_i)]} \tag{4}$$

where F_i' is the probability density function of the corresponding cumulative distribution function.

To speed up the model implementation we approximate the calculation of (4). Using (1) we can calculate that only component N_i will fail during (t_k, t_{k+1}) with probability

$$B_i(a_k) := (1 - R_i(a_k)) \prod_{j=1,\ldots,n,\ j \neq i} R_j(a_k). \tag{5}$$

Now $\sum_{i=1}^{n} B_i(a_k) + R_{\text{sys}}(a_k) < 1$, where the remainder

$$M(a_k) := 1 - \sum_{i=1}^{n} B_i(a_k) - R_{\text{sys}}(a_k)$$

describes the possibility of multiple components having their failure times at interval (t_k, t_{k+1}). This $M(a_k)$ is now divided and added to the values of $B_i(a_k)$ with respect to their sizes so that our approximation for (4) is

$$P(E_{i,k}(a_k)) \approx B_i(a_k) + \frac{B_i(a_k)}{\sum_{i=1}^{n} B_i(a_k)} M(a_k). \tag{6}$$

Since the system has a reliability requirement (3), by reliability of the system (2) we must also have

$$R_i((a_k)_i) \geq \rho \ \forall \ i \in \{1, \ldots, n\}, k \in \mathbb{N}.$$

This indicates that at each maintenance instance t_k, every component must have their reliability over the reliability threshold ρ; in other words this means at some point every component, after reaching a certain age, must be replaced. This restricts the number of possible states of the system, that is, the size of the state space S, to be finite. Since we assume that only one component can fail at each maintenance interval Δt, the number of possible failure state vectors f_k is $n + 1$. If we denote the number of possible age vectors a_k that satisfy reliability threshold (3) by h, we can write

$$|S| = h(n + 1).$$

Since there is a finite number of states, we can denote them with $\sigma_i \in S$, $i \in \{1, \ldots, |S|\}$, where every σ_i is unique.

At each maintenance interval (t_k, t_{k+1}) the system can evolve to $n + 1$ states: either all components stay functional to the end of the interval or one of the n components fails. The system will evolve from state σ_i to state σ_j during maintenance interval (t_k, t_{k+1}) with a *transition probability* P_{ij}. These probabilities are calculated with

(6) and (2). The transition probabilities P_{ij} depend only on state σ_i, so the process follows the Markov property (Puterman, 1994).

At each maintenance instance t_k, a decision is made whether one or more components in the system are maintained or not. This decision is called a *maintenance portfolio* and denoted by $p_k \subseteq N, p_k \in A$, where A is called the *action space* of the MDP. If component N_i is included in portfolio p_k, it is replaced at maintenance instance t_k. An empty maintenance portfolio indicates that no maintenance operations are carried out at the corresponding maintenance instance. After a portfolio has been applied, the state of the system is modified to represent the current state: for any replaced components N_i, both failure state $(f_k)_i$ and age $(a_k)_i$ are set to 0.

The maximum number of maintenance portfolios that can be applied in each state is 2^n. However, this number is usually smaller, as not all maintenance portfolios are feasible. Firstly, all broken components of the system must be replaced in order to restore the system into functioning condition. In addition, after applying the maintenance operations of the portfolio, reliability threshold (3) must be fulfilled. Moreover, the portfolio must take structural dependences into account: for example, in the system presented in Figure 2, if the portfolio includes chassis (C), it must also include disassembly of engines 1 and 2 (DE12). A maintenance portfolio that satisfies aforementioned conditions is called a *feasible portfolio.*

Each portfolio p_k induces a cost $c^*(p_k)$ that is sum of the costs of maintenance operations executed in p_k. This cost can be calculated from the graph that represents the structural and economical dependences of the system, as in Figure 2. Calculating $c^*(p_k)$ for a feasible portfolio requires finding a minimum-cost path from the root node of the graph to all nodes included in the portfolio p_k. This path is called the *minimum-cost absorbance tree* and it can be found using Edmond's algorithm (Kleinberg and Tardos, 2006). The total cost of a portfolio is denoted by $c(p_k)$ and it also includes the set-up cost c_0 and corrective surplus costs r. For an empty portfolio, $c(p_k) = 0$. In all other cases, the cost is given by

$$c(p_k) = c_0 + c^*(p_k) + f_k^\top r,$$

where f_k is the failure state vector of the components at t_k.

A *maintenance policy* U prescribes a rule for maintenance portfolio selection in each state s_k. Thus a policy is a function $U : S \to A$. This means that at every maintenance instance t_k, based on the state of the system, the policy U gives an unambiguous and feasible maintenance portfolio p_k. Figure 3 illustrates a Markov decision process and the relations between states, portfolios, costs and maintenance policy. A policy U is called *stationary* if it does not depend on the maintenance instance t_k: that is

$$s_k = \sigma_i \Rightarrow U(s_k) = U(\sigma_i) \quad \forall\, i \in \{1, \ldots, |S|\}, k \in \mathbb{N}.$$

The selected portfolio U has an effect on the transition probabilities between states. A *transition matrix* $P_U \in \mathbb{R}^{|S|\times|S|}$ at index (i, j) contains probabilities that the system

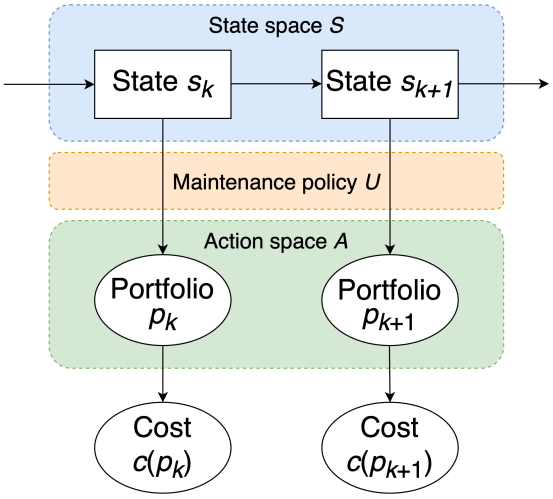

Figure 3: An illustration of a Markov decision process in maintenance optimization model.

will evolve from state σ_i to state σ_j when stationary policy U is applied. Finding the *optimal maintenance policy* for a system requires finding a stationary policy U that minimizes the long-term cumulative costs of system maintenance. In the next section, we present two algorithms for finding such policy.

3.3 Policy iteration and modified policy iteration algorithms

An optimal policy of a Markov decision process can be obtained using a policy iteration algorithm (PI; see Howard, 1960). However, since the policy iteration algorithm is computationally somewhat heavy, Puterman and Shin (1978) present a modified policy iteration algorithm (MPI), that can be used to find an ε-optimal policy with $\varepsilon > 0$. In this thesis, we examine a *discounted* Markov decision process, where future costs are discounted with a *discount factor* of $\lambda \in [0, 1)$.

We first look at the policy iteration algorithm. We can calculate the expected discounted cost of stationary policy U over k maintenance instances with starting state σ_i by

$$V_k(\sigma_i, U) = c(U(\sigma_i)) + \lambda \sum_{j=1}^{|S|} (P_U)_{ij} \cdot V_{k-1}(\sigma_j, U), \tag{7}$$

which is a recursive formulation of the Bellman equation (Bellman, 1957). If we denote $v_k(U) \in \mathbb{R}^{|S|}$ such that $(v_k(U))_i = V_k(\sigma_i, U)$ for all $i \in \{1, \ldots, |S|\}$, we can write (7) as

$$v_k(U) = c_U + \lambda P_U v_{k-1}(U), \tag{8}$$

where c_U is a *cost vector* for policy U; that is, $(c_U)_i = c(U(\sigma_i))$ for all $i \in \{1, \ldots, |S|\}$. Now the expected total discounted cost of policy U can be calculated by letting $k \to \infty$, so that from (8) we get

$$v(U) = c_U + \lambda P_U v(U),$$

which can be rewritten as

$$(I - \lambda P_U)v(U) = c_U.$$

For $0 \leq \lambda < 1$ it can be shown, that the matrix $I - \lambda P_U$ is inversible (Puterman, 1994). Thus the total expected discounted cost of a policy is

$$v(U) = (I - \lambda P_U)^{-1} c_U. \tag{9}$$

The optimal policy U^* is a solution to a minimizing problem

$$v(U^*) = \min_U \{c_U + \lambda P_U v(U)\}. \tag{10}$$

The system of equations of form (10) is called the *optimality equations* of the MDP. This problem can be solved iteratively using policy iteration algorithm, that in Puterman (1994) is presented as follows:

Step 1. Set $n = 0$ and select an arbitrary policy U_0.

Step 2. (Policy evaluation) Obtain $v(U_n)$ by solving

$$v(U_n) = (I - \lambda P_{U_n})^{-1} c_{U_n}.$$

Step 3. (Policy improvement) Choose U_{n+1} to satisfy

$$U_{n+1} \in \arg\min_U \{c_U + \lambda P_U v(U_n)\}.$$

Step 4. If $U_{n+1} = U_n$, stop and set $U^* = U_n$. Otherwise increment n by 1 and return to step 2.

If the size of the state space S is finite, and for each state, the number of feasible portfolios is also finite, the policy iteration algorithm is guaranteed to terminate in a finite number of iterations (Puterman, 1994).

Obtaining the precise solution to the system of linear equations in step 2 of the policy iteration algorithm is not necessary in order to find the optimal policy. Alternatively, we can approximate the value for $v(U_{n+1})$ by approaching it from the approximated value of $v(U_n)$. Thus we can avoid solving the system of linear equations in step 2 by using a modified policy iteration algorithm. The algorithm finds an ε-optimal policy denoted by U_ε. The MPI algorithm is adopted from Puterman (1994) as follows:

Step 1. Set $n = 0$, select an arbitrary policy U_0, set $v(U_0) = c_{U_0}$, specify $\varepsilon > 0$ and select a sequence of non-negative integers $\{m_n\}$.

Step 2. (Policy improvement) Choose U_{n+1} to satisfy

$$U_{n+1} \in \arg\min_U \{c_U + \lambda P_U v(U_n)\}$$

setting $U_{n+1} = U_n$ if possible (when $n > 0$).

Step 3. (Partial policy evaluation).

(a) Set $k = 0$ and select $u^0(U_n) \in \mathbb{R}^{|S|}$ by

$$u^0(U_n) := \min_U \{c_U + \lambda P_U v(U_n)\}.$$

(b) If $||u^0(U_n) - v(U_n)|| < \varepsilon(1-\lambda)/2\lambda$, go to step 4. Otherwise go to (c).

(c) If $k = m_n$, go to (e). Otherwise compute $u^{k+1}(U_n)$ by

$$u^{k+1}(U_n) = c_{U_{n+1}} + \lambda P_{U_{n+1}} u^k(U_n).$$

(d) Increment k by 1 and return to (c).

(e) Set $v(U_{n+1}) = u^{m_n}(U_n)$, increment n by 1 and go to step 2.

Step 4. Set $U_\varepsilon = U_{n+1}$ and stop.

The sequence $\{m_n\}$ may be, for example, a constant number for all iterations, a prefixed pattern of non-negative integers, or selected adaptively by requiring

$$||u^{m_n+1}(U_n) - u^{m_n}(U_n)|| < \varepsilon_n$$

for some ε_n. Similarly to the PI algorithm, the MPI algorithm is guaranteed to terminate in a finite number of iterations for all sequences $\{m_n\}$. However, the number of iterations the algorithm takes to converge depends on the selection of $\{m_n\}$, which is covered in Section 4.2.

An implementation of the PI algorithm is done by Leppinen (2020), so in this thesis we focus on the implementation of the MPI algorithm. As the performance of an algorithm usually depends on the implementation, next we look at some of the aspects of the implementation of the MPI algorithm in MATLAB.

3.4 Implementation of the MPI algorithm in MATLAB

In this thesis, the modified policy iteration algorithm is implemented in MATLAB as a function, similarly as the policy iteration algorithm is implemented by Leppinen (2020). This way we can straightforwardly call the algorithms with same parameters and compare the results and running times of the algorithms.

The first step in the MATLAB function is to define the Markov decision process in an initialization step. In the initialization step the dependences of the maintenance optimization model are formed, and the costs of different portfolios are calculated using Edmond's algorithm. In addition, the reliabilities of the components at each age a_k are calculated using (1), and the allowed age combinations of the components are selected using the system reliability requirement (3). Thus we have the unique states σ_i, and feasibility of each portfolio for every state can be tested. The initial

policy U_0 in step 1 of the MPI algorithm is selected by choosing the cheapest feasible portfolio for each state σ_i, after which transition probabilities $(P_{U_0})_{ij}$ are calculated.

In general, in the implementation we can assume that $|S| \gg n$. Since the system can evolve from state σ_i to $n+1$ states and the dimension of the transition matrix P_U is $|S| \times |S|$, P_U is very sparse. If we order the states σ_j that can follow state σ_i such that σ_j denotes the failure of component N_1, σ_{j+1} denotes the failure of N_2 etc., we can save a considerable amount of memory by storing the probabilities in a matrix of dimension $|S| \times (n+2)$, where only the non-zero probabilities $(P_U)_{ij}$ and the starting index j are stored for each state σ_i. This also allows speeding up the matrix-vector multiplications in steps 2 and 3(c) of the MPI algorithm, since the multiplications with 0 can be omitted.

For the sake of simplicity, the sequence $\{m_n\}$ in the MPI algorithm is selected to be a constant value $m_n = m$ for all n. The stopping criterion in step 3(b) of the MPI algorithm is implemented using maximum norm, since it is computationally less heavy and in tests produces similar results as the Euclidean norm.

4 Results

In this section we test the performance of the modified policy iteration and policy iteration algorithms. First in Sections 4.1 and 4.2 we define an example system and its parameters used in the tests. Later in Sections 4.3 and 4.4 we present the performance results for the algorithms.

The tests performed in this section are run with a workstation computer having Intel Core i5-11600 processor at 2.80 GHz clock speed, 32 GB of RAM and running MATLAB R2022a on Ubuntu.

4.1 Example system

The MPI and PI algorithms can be tested using an imaginary example system similar to what is presented in Figure 2. The system is a transportation system, and since measuring the usage of such system is customary to do by mileage, the maintenance interval between each maintenance instance t_k is assumed to be 100 000 kilometers. In the implementation of the algorithm, the maintenance interval is scaled to be $\Delta t = 1$ for the 100 000 km maintenance interval.

The failure distributions for the components are chosen to be Weibull distributions, because those are common in reliability analysis literature (e.g. Kapur and Lamberson, 1977). The probability density function of a Weibull distributed random variable t is given by

$$F'(t;\gamma,k) = \begin{cases} \frac{k}{\gamma}\left(\frac{t}{\gamma}\right)^{k-1} e^{-(t/\gamma)^k} & t \geq 0, \\ 0 & t < 0, \end{cases} \tag{11}$$

where $\gamma > 0$ and $k > 0$ are called the shape and scale parameters of the distribution, respectively. In order to fulfil the assumption of increasing failure rate, we set $k > 1$.

Component	Weibull parameters k	γ	Component spesific costs Dismantle	Replacement	Corrective surplus
Engine 1	5.1	10.8	23	393	300
Engine 2	5.1	10.8	28	403	300
Chassis	5.5	9.9	167	413	160
Wheels	4.0	9.0	0	1000	613

Table 1: Weibull distribution parameters and component specific costs of the example system.

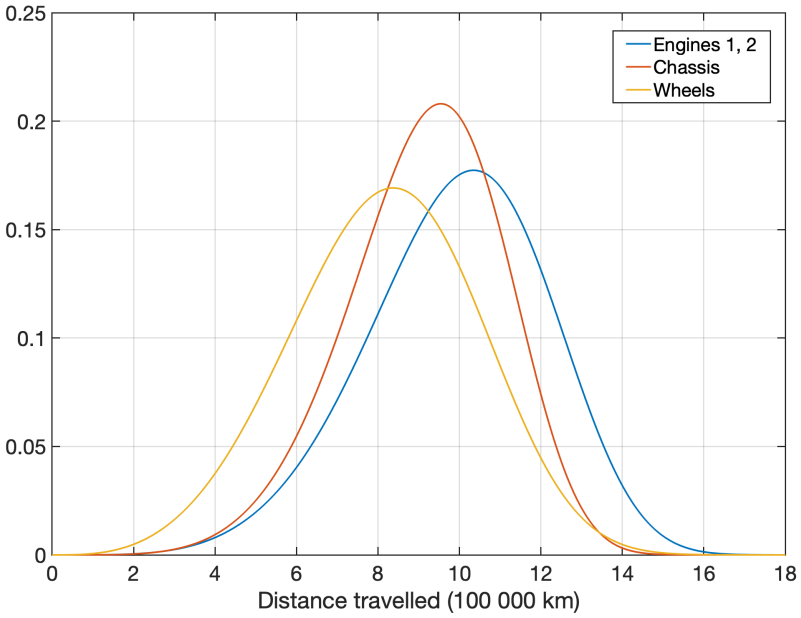

Figure 4: Failure probability density functions of the components of the example system.

The cumulative distribution function of (11) is given by

$$F(t;\gamma,k) = \begin{cases} 1 - e^{-(t/\gamma)^k} & t \geq 0, \\ 0 & t < 0. \end{cases}$$

The Weibull distribution parameters for the components in the example system are presented in Table 1. The probability density functions of the Weibull distributions are plotted in Figure 4. Table 1 also shows the component specific costs associated with the system. In addition, the maintenance set-up cost is $c_0 = 388$.

4.2 Selection of parameters

The number of iterations in steps 3(b)–3(d) of the MPI algorithm is selected by setting the sequence $\{m_n\}$ to be a constant m for every n. The constant m can be

	$\lambda = 0.98$		$\lambda = 0.99$		$\lambda = 0.993$	
m	N_{iter}	T_{alg} (s)	N_{iter}	T_{alg} (s)	N_{iter}	T_{alg} (s)
5	37	3.90	74	7.85	94	10.00
10	20	2.76	39	5.45	55	7.73
15	14	2.38	28	4.85	39	6.81
20	11	2.19	21	4.32	31	6.43
30	**8**	**2.05**	14	3.73	23	6.29
40	7	2.20	**11**	**3.58**	18	6.07
50	7	2.62	10	3.88	**12**	**4.73**
75	7	3.68	8	4.27	10	5.46
100	7	4.75	8	5.49	8	5.50

Table 2: The number of iterations and computation time before convergence with different numbers of MPI algorithm inner loops and different discount factors λ.

chosen by performing tests with a different number of iterations and measuring their convergence times. The test is run using discount factors $\lambda \in \{0.98, 0.99, 0.993\}$, reliability threshold $\rho = 0.9$ and maintenance interval $\Delta t = 1$, which results to a state space of size $|S| = 6840$. In addition, the stopping criterion in step 3(b) of the MPI algorithm is set to $\varepsilon = 0.01$ for every test. The results of the test runs are shown in Table 2. In the table, m denotes the number of iterations in steps 3(b)–3(d), N_{iter} denotes the number of iterations that the algorithm takes before convergence and T_{alg} denotes the algorithm convergence time in seconds. Time T_{alg} does not include the time it takes to initialize the problem, as described in Section 3.4, so it only measures the time the algorithm takes to converge to an optimal policy.

Based on the results presented in Table 2, it seems that a larger number of approximation step iterations leads to a smaller number of overall algorithm iterations; however, the algorithm run time has a minimal value at some point. In these tests the smallest convergence time seems to occur around $30 \leq m \leq 50$. Based on this test, the sequence $\{m_n\}$ was selected to be 40 for all n.

4.3 Comparison of PI and MPI algorithms

The performance of both policy iteration and modified policy iteration algorithms are tested by measuring their convergence times while varying parameter values of the example system described in Section 4.1. Furthermore, the optimal policies found by the algorithms are tested for similarity.

An important detail in the implementation of the PI algorithm by Leppinen (2020) is that it uses MATLAB implementation of conjugate gradients squared method for solving the system of linear equations in step 2 of the PI algorithm. This makes the PI implementation considerably faster than by solving the system with the standard method of using the backslash operator.

First, the performances of the algorithms are tested with different reliability thresholds

		MPI		PI	
ρ	$\lvert S\rvert$	N_{iter}	T_{alg} (s)	N_{iter}	T_{alg} (s)
0.999	40	24	0.05	1	0.06
0.99	550	18	0.45	4	0.09
0.98	1225	16	0.89	7	0.12
0.96	2560	14	1.63	9	0.94
0.93	4780	13	2.83	9	3.70
0.90	6840	11	3.38	8	6.06
0.85	10570	13	6.31	9	19.79
0.80	15520	9	6.26	10	38.06
0.75	19750	11	9.94	9	65.86
0.70	25060	10	11.30	8	121.38

Table 3: Convergence times of MPI and PI algorithms with different reliability threshold levels.

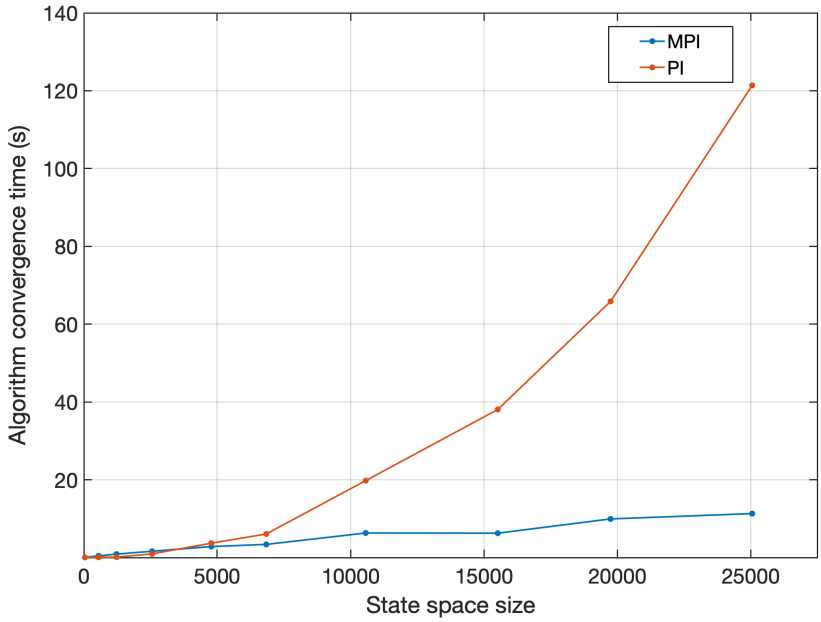

Figure 5: Algorithm convergence times of MPI and PI algorithms for different state spaces sizes.

ρ. The reliability threshold ρ affects the state space size $|S|$: if a lower reliability is allowed for the system, there are more possible age combinations for the components. Since a smaller ρ value leads to a larger state space size while not changing any other parameters of the model, the test gives a benchmark of the algorithms' overall performance with differently sized state spaces. The test is run using a discount factor of $\lambda = 0.99$, and the results are presented in Table 3. In the table N_{iter} denotes the number of algorithm iterations, and T_{alg} denotes the algorithm convergence time in seconds. The convergence times are also plotted in Figure 5.

From Table 3 and Figure 5 we see that with small state spaces the PI algorithm seems to perform better. However, as the size of the state space grows, MPI algorithm can find the optimal policy much faster than PI algorithm: for example, with $|S| = 25060$ the MPI algorithm converges over 10 times faster than PI algorithm. The overall number of iterations before convergence is generally higher for the MPI algorithm, which indicates that selecting the policy with an approximated value for the discounted cost of the policy $v(U)$ is less optimal than using the exact solution for (9). On the other hand, each iteration of the MPI algorithm takes much less time than on PI algorithm.

The convergence time of the algorithms are also tested with different discount factors λ. The discount factor does not affect the size of the state space S, so each test is run using reliability threshold $\rho = 0.9$, which results in $|S| = 6840$. The results of the tests are presented in Table 4 and Figure 6. From the table we see that as discount factor λ approaches 1, MPI algorithm needs a larger number of iterations to converge, which results as a longer computation time. This is natural, since the termination criterion in step 3(b) of the MPI algorithm becomes increasingly constraining as λ grows. Conversely, PI algorithm does not seem to be that sensitive for the value of the discount factor, which is in line with results of Leppinen (2020).

In every comparison test for MPI and PI algorithms, the policies found by the algorithms are compared for similarity. In every test case presented in Tables 3 and 4, the optimal policy U^* and the ε-optimal policy U_ε found by the algorithms are exactly the same. This indicates that the chosen accuracy of $\varepsilon = 0.01$ is sufficiently small.

	MPI		PI	
λ	N_{iter}	T_{alg} (s)	N_{iter}	T_{alg} (s)
0.90	6	1.78	7	4.41
0.93	7	2.11	9	4.42
0.95	7	2.08	8	4.03
0.97	7	2.07	9	5.36
0.98	7	2.09	8	5.31
0.99	11	3.42	8	5.91
0.993	16	5.13	10	6.71
0.995	23	7.49	9	6.28
0.998	35	11.47	9	6.50
0.999	92	30.55	7	5.12

Table 4: Number of iterations and convergence times of MPI and PI algorithms with different discount factors.

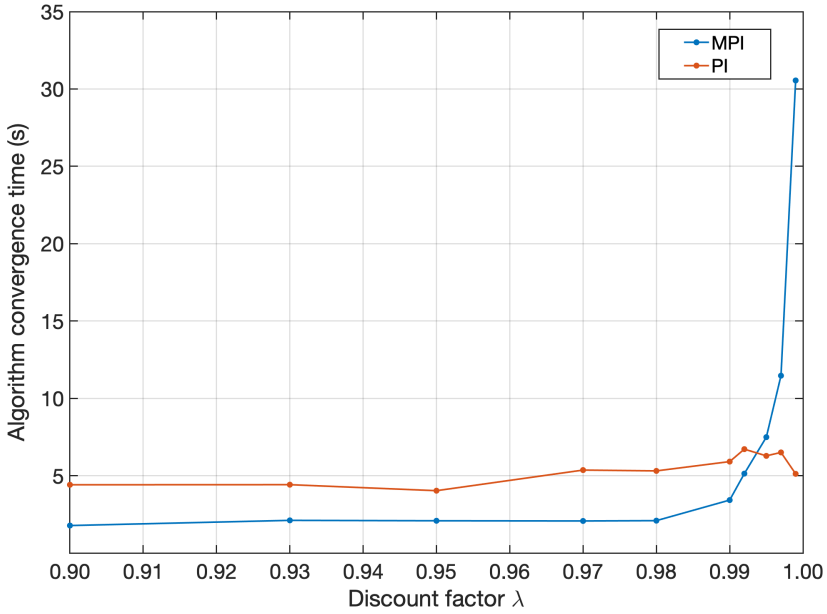

Figure 6: Algorithm convergence times of MPI and PI algorithms for different discount factors.

4.4 Performance of MPI algorithm in systems with larger state spaces

While comparing MPI and PI algorithms with varying state space size, it quickly becomes apparent that the implementation of PI algorithm requires more memory than is available on the test computer. For example, if state space size is $|S| = 75000$, storing the transition matrix $P_U \in \mathbb{R}^{|S|\times|S|}$ as double precision floating point numbers requires about 45 gigabytes of memory, whereas storing the matrix as described in section 3.4 only requires about 3.6 megabytes.

The maintenance interval has a great effect on the size of the state space S: if the maintenance interval is shorter, then there are more possible age combinations for the components. On the other hand, longer maintenance interval leads to smaller reliability of the system and fewer possible component age combinations. For example, in the example system with $\rho = 0.9$, halving the maintenance interval increases the state space size $|S|$ from 6840 to 232755. Thus these tests with varying maintenance interval are only performed with the MPI algorithm.

Changing the maintenance interval also has an effect on the discount factor. If the discount factor is assumed to be $\lambda = 0.99$ for the span of 100000 kilometers, when varying maintenance interval we must also calculate a new discount factor. The discount factor can be calculated as

$$\lambda = 0.99^{\Delta t},$$

where $\Delta t = 1$ for a maintenance interval of 100000 km.

The MPI algorithm is now tested with different maintenance intervals and the results are presented in Table 5. In the table, also the time for the algorithm initialization step explained in Section 3.4 is presented as T_{init}. The results are also plotted in Figure 7. From the results we see that the convergence time for MPI algorithm seems to grow almost linearly as the state space grows. Moreover, we see that the MPI algorithm can be used to solve systems with much larger state spaces than PI algorithm, as storing the transition matrix P_U for maintenance interval $\Delta t = 0.5$ would require about 430 gigabytes of memory.

Δt	λ	$\lvert S\rvert$	N_{iter}	T_{init} (s)	T_{alg} (s)
1.0	0.99	6840	11	0.98	3.63
0.95	0.9905	9090	16	0.65	7.07
0.90	0.9910	11635	17	0.81	9.72
0.85	0.9915	15875	10	1.06	7.44
0.80	0.9920	21600	17	1.42	17.79
0.75	0.9925	29885	13	1.95	18.52
0.70	0.9930	42185	13	2.74	26.26
0.65	0.9935	61890	15	3.99	45.07
0.60	0.9940	92875	12	6.03	53.27
0.55	0.9945	143040	12	9.42	81.86
0.50	0.9950	232755	13	15.92	145.43

Table 5: Convergence time of MPI algorithm for different maintenance intervals.

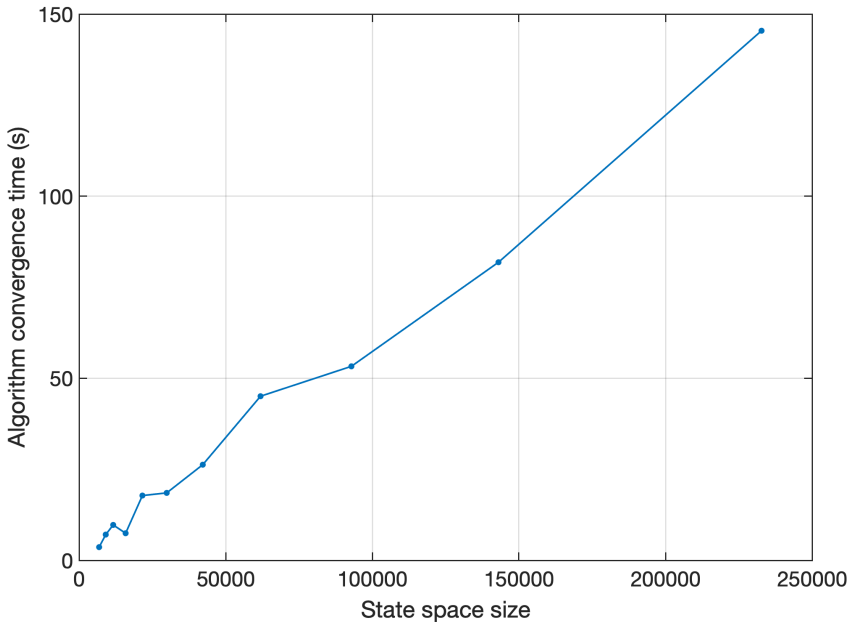

Figure 7: Algorithm convergence time of MPI algorithm for different state space sizes when varying the maintenance interval length.

5 Conclusion

In this thesis we presented a maintenance optimization model that utilizes a discrete time discounted Markov decision process to model the state and maintenance decisions of a multi-component system with structural and economical dependences. In addition, we presented and implemented modified policy iteration algorithm, that can be used to find an optimal maintenance policy for the system, and compared the computational performance of the modified policy iteration and policy iteration algorithms.

Based on the results presented in Section 4, the modified policy iteration algorithm seems to be a viable method for solving the optimal policy for a Markov decision process in the maintenance optimization model. The MPI algorithm has a significant speed advantage compared to the ordinary policy iteration algorithm, while in every test case the algorithms found the same optimal policy. However, when applying the MPI algorithm, some caution is required since at some specific cases an approximative algorithm takes considerably longer to converge.

One of the most important benefits of the MPI algorithm implemented in this thesis is the memory save explained in Section 3.4. While the usual implementations of matrix inversion algorithms require a square matrix input, the ability to store a transition matrix of the MPD into a more compact data structure allows solving optimal policies for much larger systems than with ordinary PI. Since adding components to the system model increases the size of the state space greatly, MPI algorithm might be usable for solving maintenance policies for much larger systems.

At the moment, the algorithm uses only one computer processor (CPU) thread to perform calculations. However, the large matrix-vector multiplications executed in the MPI algorithm should be reasonably straightforward to parallelize to multiple CPU cores, which would propose an interesting topic for future development. In addition, the maintenance scheduling model and MPI algorithm could be tested for systems containing more components and larger state space.

The maintenance optimization model presented in this thesis could also be further developed by making it model real technical system more accurately: for example, more component condition states instead of just two, working or not working, could be included. In addition, incorporation of resource dependence, for example in form of limited available repair time per maintenance instance, and its effects on feasible maintenance policies could propose an interesting research subject. Such model could also include a cost for the system shutdown time.

References

R. E. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

M. Ben-Daya, U. Kumar, and D. N. Prabhakar Murthy. *Introduction to maintenance engineering: modelling, optimization and management.* John Wiley & Sons, 2016.

M. Bevilacqua and M. Braglia. The analytic hierarchy process applied to maintenance strategy selection. *Reliability Engineering & System Safety*, 70:71–83, 2000.

B. de Jonge and P. A. Scarf. A review on maintenance optimization. *European Journal of Operational Research*, 285:805–824, 2020.

B. de Jonge, W. Klingenberg, R. Teunter, and T. Tinga. Optimum maintenance strategy under uncertainty in the lifetime distribution. *Reliability Engineering & System Safety*, 133:59–67, 2015.

R. Dekker. Applications of maintenance optimization models: a review and analysis. *Reliability Engineering & System Safety*, 51:229–240, 1996.

R. A. Howard. *Dynamic programming and markov processes.* John Wiley & Sons, 1960.

K. C. Kapur and L. R. Lamberson. *Reliability in engineering design.* John Wiley & Sons, 1977.

J. Kleinberg and E. Tardos. *Algorithm design.* Pearson Education India, 2006.

J. Leppinen. A dynamic optimization model for maintenance scheduling of a multi-component system. Master's thesis, Aalto University School of Science, 2020.

L. M. Maillart. Maintenance policies for systems with condition monitoring and obvious failures. *IIE Transactions*, 38:463–475, 2006.

M. Olde Keizer, S. Flapper, and R. Teunter. Condition-based maintenance policies for systems with multiple dependent components: A review. *European Journal of Operational Research*, 261:405–420, 2017.

M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, 1994.

M. L. Puterman and M. C. Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24:1127–1137, 1978.

L. C. Thomas. A survey of maintenance and replacement models for maintainability and reliability of multi-item systems. *Reliability Engineering*, 16:297–309, 1986.