A policy iteration algorithm for Markov decision processes skip-free in one direction

J. Lambert, B. Van Houdt and C. Blondia
University of Antwerp, Department of Mathematics and Computer Science,
Performance Analysis of Telecommunication Systems Research Group,
Middelheimlaan, 1, B-2020 Antwerp - Belgium,
{joke.lambert,benny.vanhoudt,chris.blondia}@ua.ac.be

ABSTRACT

In this paper we present a new algorithm for policy iteration for Markov decision processes (MDP) skip-free in one direction. This algorithm, which is based on matrix analytic methods, is in the same spirit as the algorithm of White (Stochastic Models, 21:785-797, 2005) which was limited to matrices that are skip-free in both directions.
Optimization problems that can be solved using Markov decision processes arise in the domain of optical buffers, when trying to improve loss rates of fibre delay line (FDL) buffers. Based on the analysis of such an FDL buffer we present a comparative study between the different techniques available to solve an MDP. The results illustrate that the exploitation of the structure of the transition matrices places us in a position to deal with larger systems, while reducing the computation times.

Categories and Subject Descriptors

G.3 [**Probability and Statistics**]: Markov processes; G.1.6 [**Numerical Analysis**]: Optimization—*Global optimization*

General Terms

Policy iteration algorithm, Performance

Keywords

Matrix analytic methods, Markov decision process, skip-free in one direction, optical buffer, fibre delay lines, loss rate.

1. INTRODUCTION

Markov decision processes [1] date back to the 1950s and provide a mathematical framework for studying a wide range of optimization problems. Typically, the optimization problem is, among others, characterized by a set of states $\mathcal{S}$ and a set of actions $\mathcal{A}$. Each action $a \in \mathcal{A}$ can be performed from any state $h \in \mathcal{S}$ and causes a transition to a new state $h' \in \mathcal{S}$, at some cost $c_h(a)$. As such, we have a transition matrix $P(a)$ for each action $a \in \mathcal{A}$. The question is now: which action $a \in \mathcal{A}$ should we choose when in state h, for each $h \in \mathcal{S}$, such that the long-run average cost is minimal? In other words, what is the optimal *policy* which maps $\mathcal{S}$ to $\mathcal{A}$ such that the costs are minimized.

Two important iterative approaches that determine the optimal policy are the *policy* iteration and the *value* iteration algorithm. Policy iteration has the advantage that it is guaranteed to converge in a finite number of steps (typically, between 3 and 15 iterations [9]), but requires the solution of a linear system of equations at each iteration. The value iteration algorithm is faster per iteration as it avoids solving linear systems, but in general requires significantly more iterations.

In case the matrices $P(a)$ are structured, policy iteration can potentially also exploit this structure to reduce the amount of computation time and memory at each iteration. Within the matrix analytic paradigm, this was first demonstrated by White [10] in case the matrices $P(a)$ are skip free in both directions, meaning the state space S is partitioned into $M+1$ sets, labeled S_0 to S_M, and a transition from a state $h \in S_i$ to h' can only occur if $h' \in S_{i-1} \cup S_i \cup S_{i+1}$.

The result by White will be generalized in this paper by developing an algorithm for the case where the $P(a)$ matrices are skip-free in only one direction. Our algorithm bears the same resemblance with the linear reduction algorithm in [7] to determine the steady state vector of a Markov chain skip-free in one direction, as White's algorithm does with [8]. However, when the chain is only skip-free in one direction, one can apply a reduction from the right-to-left or left-to-right which leads to two somewhat different algorithms. We will explore both possibilities and argue that the right-to-left approach seems to allow a bigger gain in terms of memory usage and computation time.

MDPs have a wide range of application areas. Our motivation lies in understanding the behavior of optical buffers, called fibre delay lines (FDLs) present in future optical communication networks [2], [4], [3] and [6]. Unlike electronic random-access memory (RAM), FDLs can only delay packets, called optical bursts (OB), by a discrete set of values. In a slotted *equidistant* system typical delay values are $0, D, 2D, 3D, \ldots, ND$ time slots for some integers N and D, where the main performance measure of such an FDL buffer is its loss rate.

To improve the loss rate of the equidistant FDL buffer, we devised a new mechanism in [5] called the preventive drop mechanism (see Section 4 for details). As demonstrated in [5], the optimal policy for the preventive drop mechanism

SMCtools '07, October 26, 2007, Nantes, France

can be determined via a Markov Decision Process. In [5] the value iteration algorithm was used for this purpose, causing slow convergence to the optimal policy. Also, when trace-based input was used for the OB length distribution, some clustering of this distribution was required to reduce the computation times (and the size of the state space $\mathcal{S}$). Using the new approach, we are now in a position to tackle larger systems, while reducing the computation times. We have used the new approach to validate the approximation made in [5] by clustering the OB length distribution.

The paper is structured as follows. A general introduction on skip-free in one direction Markov chains and on Markov decision processes is given in Section 2. Section 3 describes the new policy iteration algorithm for skip-free in one direction MDPs. The general notion of FDL buffers is introduced in Section 4 and we show how to capture its behavior by a Markov decision process. In Section 5 we present a comparative study between the different techniques available to solve an MDP and we elaborate on a numerical example.

2. MARKOV DECISION PROCESSES SKIP-FREE IN ONE DIRECTION

In this section we will describe the structure of a Markov Decision process (MDP) skip-free in one direction. To do so, we start by giving a short introduction on skip-free in one direction Markov chains.

2.1 Skip-free in one direction Markov chains

A Markov chain skip-free in one direction is a discrete-time Markov chain defined on a finite state space $\mathcal{S} = \{k \mid k = 1, \dots K\}$, i.e., at each time $t \geq 0, X(t) = k \in \{1, \dots, K\}$. The state space $\mathcal{S}$ is partitioned into $M+1$ sets S_i, for $i = 0, \dots, M$, where $|S_i| = b_i$. Further, the transition probability matrix P of this chain has a block skip-free (to the left) structure:

$$P = \begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & \dots & A_{0,M} \\ A_{1,0} & A_{1,1} & A_{1,2} & \dots & A_{1,M} \\ 0 & A_{2,1} & A_{2,2} & \dots & A_{2,M} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & A_{M,M-1} & A_{M,M} \end{bmatrix}. \quad (1)$$

The matrix P is of size $K \times K$, where the subblock $A_{i,k}$ of size $b_i \times b_k$ holds the transition probabilities between the states of the set S_i and S_k. The stationary probability vector $x = (x_0, \dots, x_M)$, where x_i has length b_i, satisfies:

$$x = xP \qquad xe = 1,$$

where e is a column vector with all entries equal to one. Within this paper we will restrict ourselves to the case where $b_0 = b_1 = \dots = b_M = b = K/(M+1)$, meaning all subblocks are square and have the same dimension. The ideas introduced in this paper can be generalized without much difficulty to the general setting.

2.2 Markov decision process skip-free in one direction

For each state h, there exists a set $\mathcal{A}(h)$ of decisions or actions. In our case the set $\mathcal{A}(h)$ is the same for all h; hence we simply denote this set as $\mathcal{A}$. Each action incurs an immediate cost and also affects the probability law for the next transition. A formal definition of the MDP is given by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C} \rangle$, where $\mathcal{S}$ is the set of possible states, $\mathcal{A}$ is the set of possible actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is the state transition function specifying the probability $\mathcal{P}\{h' \mid h, a\} = p_{h,h'}(a)$ of observing a transition to state $h' \in \mathcal{S}$ after taking action $a \in \mathcal{A}$ in state $h \in \mathcal{S}$ and, finally $\mathcal{C} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a function specifying the cost $c_h(a)$ of taking action $a \in \mathcal{A}$ at state $h \in \mathcal{S}$ [9]. Notice, the cost $c_h(a)$ depends on the action a taken in state h, but not on the state h' visited (as a result of taking action a in state h). Thus, if at a decision moment the action a is chosen in state h, then regardless of the past history of the system, the following happens: (i) An immediate cost $c_h(a)$ is incurred, (ii) At the next decision moment the system will be in state h' with probability $p_{h,h'}(a)$.

The goal of the decision model is to prescribe a policy R for controlling the system, such that the cost is minimal. Formally, a policy R is a mapping $R : \mathcal{S} \to \mathcal{A}$ and under a given policy R, action $R(h)$ is always executed whenever we visit state h.

For a given policy R we can define the long-run average cost from state h as follows:

$$J_R(h) = E\left[\sum_{t=0}^{\infty} \alpha^t c_{X(t)}(R(X(t))) \,\middle|\, X(0) = h\right], \quad (2)$$

where $0 < \alpha < 1$ is the discount factor. An optimal policy R_{opt} is defined to be a policy which realises the minimum long-run average cost $J_R(h)$ over all policies R and this for *all* initial states $h \in \mathcal{S}$. It is well-known that the optimal (minimum) long-run average cost is a solution of

$$\begin{aligned} J^*(h) &= \min_R J_R(h) \\ &= \min_{a \in \mathcal{A}} E\left[c_h(a) + \alpha J^*(X(1)) \mid X(0) = h\right] \\ &= \min_{a \in \mathcal{A}} \left(\sum_{k=1}^{K} p_{h,k}(a)(c_h(a) + \alpha J^*(k)) \right), \end{aligned} \quad (3)$$

for $h = 1, \dots, K$ and where $p_{h,k}(a)$ denotes the transition probability from state h to state k under action a. With some abuse of notation, in vector matrix form this becomes

$$J^* = \min_a (c(a) + \alpha P(a) J^*), \quad (4)$$

with the vector $J^* = (J^*(1), J^*(2), \dots, J^*(K))$, entry h of the column vector $c(a)$, denoted as $[c(a)]_h$, equal to $c_h(a)$ and $P(a)$ the transition matrix given that action a is executed. Equation (4) is a set of non-lineair equations that in general cannot be solved directly. For a given policy R, the associated long-run average cost satisfies the equation (see [1]):

$$J_R(h) = \sum_{k=1}^{K} p_{h,k}(R(h))(c_h(R(h)) + \alpha J_R(k)), \quad (5)$$

or in vector form

$$J_R = c(a) + \alpha P(a) J_R, \quad (6)$$

where a is determined by the policy R for each component $h \in \mathcal{S}$. The policy iteration algorithm for determining the optimal long-run average cost J^* works as follows [1]:

- *Step 0:* Select an initial policy R_0 and set $n = 0$. This leads to an initial set of actions $R_0(h)$ for $h = 1, \dots, K$.
- *Step 1:* We then evaluate the policy R_n by solving (6) to yield J_{R_n}. This step is called policy evaluation.

- *Step 2:* Afterwards we update the policy to R_{n+1} by using

$$R_{n+1}(h) = \text{argmin}_{a\in\mathcal{A}}\{c_h(a) + \sum_{k=1}^{K} p_{h,k}(a)J_{R_n}(k)\},$$

where the minimum is taken over all actions $a \in \mathcal{A}$ for all $h \in \mathcal{S}$. Notice, this is equivalent to saying that R_{n+1} minimizes $c(R(h))+P(R(h))J_{R_n}$ over all policies for all $h \in \mathcal{S}$. This step is called policy improvement.

- *Step 3:* The algorithm is stopped, returning the desired policy $R_{\text{opt}} = R_n$, when $R_n = R_{n-1}$, as policy iteration guarantees convergence to the optimal policy in a finite number of steps.

A Markov decision process, characterized by the matrices $P(a)$, for $a \in \mathcal{A}$ is skip-free in one direction (to the left) if all the matrices $P(a)$ are skip-free (to the left) and consequently, all transition matrices corresponding to any policy R are skip-free.

3. MATRIX ANALYTIC TECHNIQUES FOR POLICY EVALUATION

In this section we only study the policy evaluation step as it is the most expensive step. We drop all explicit references to the policy R under evaluation to make the notations easier. Thus, we will write $c(h)$ for the cost $c_h(R(h))$ from state h under the chosen policy R and $p_{h,k}$ denotes the transition probability $p_{h,k}(R(h))$ from state h to state k under the chosen policy, where P is composed of subblocks $A_{m,n}$ as defined in (1). Let us denote the corresponding long-run average cost row vector by $J^T = (J_0^T, J_1^T, \ldots, J_M^T)$ where ^{T} denotes the transpose and each J_m is a column vector of size b. The matrix equation (6) can be written as the set of equations:

$$\begin{aligned} J_0 &= c_0 + \sum_{i=0}^{M}(\alpha A_{0,i}J_i) \\ J_m &= c_m + \sum_{i=m-1}^{M}(\alpha A_{m,i}J_i) \\ J_M &= c_M + \sum_{i=M-1}^{M}(\alpha A_{M,i}J_i), \end{aligned} \tag{7}$$

for $m = 1, \ldots, M-1$ and where the size b column vector c_i equals $(c(ib+1), \ldots, c((i+1)b))^T$, for $i = 0, \ldots, M$.

Two different approaches to solve this system of equations efficiently are discussed next. Both may be regarded as generalizations of the approach developed by White [10] for the Quasi-Birth-Death case, where the chain is skip-free in both directions. Both approaches are also closely related with the algorithm used to compute the stationary vector of a Markov chain skip-free in one direction developed by Latouche, Jacobs and Gaver [7], in the same manner as the approach taken by White relates to the linear reduction algorithm for the stationary vector of a finite QBD discussed in [8]. The two approaches differ in the way the lineair reduction is performed: either from right-to-left or from left-to-right.

3.1 A policy evaluation algorithm for MDPs skip-free in one direction: the right-to-left approach

Analogue to the lineair reduction method for solving skip-free in one direction Markov chains [7], we start by defining the matrices $\bar{A}_m$ and $\Theta_{k,m}$ that turn out to be useful to evaluate the long-run average cost $J(h)$. The matrices $\bar{A}_m$ with $0 \le m \le M$ and $\Theta_{k,m}$ with $0 \le k < m \le M$ are recursively determined as follows:

$$\bar{A}_M = \alpha A_{M,M}, \tag{8a}$$

$$\bar{A}_m = \alpha(A_{m,m} + \Theta_{m,m+1}(I - \bar{A}_{m+1})^{-1}A_{m+1,m}), \tag{8b}$$

for $m = M-1, M-2, \ldots, 0$ and

$$\Theta_{k,M} = \alpha A_{k,M}, \tag{9}$$

with $k = 0, \ldots, M-1$ and for $m = M-1, M-2, \ldots, 1$ we set

$$\Theta_{k,m} = \alpha(A_{k,m} + \Theta_{k,m+1}(I - \bar{A}_{m+1})^{-1}A_{m+1,m}), \tag{10}$$

where $k = 0, \ldots, m-1$. Besides we also define a vector $\bar{c}_m$, for $0 \le m \le M$ as follows:

$$\bar{c}_M = c_M \tag{11a}$$

$$\bar{c}_m = c_m + \sum_{j=m+1}^{M}\left(\Theta_{m,j}(I - \bar{A}_j)^{-1}\bar{c}_j\right), \tag{11b}$$

where $m = M-1, \ldots, 0$.
Let us now use these matrices and vectors to rewrite equation (7). Consider the last equation in (7). We can write:

$$\begin{aligned} J_M &= (I - \alpha A_{M,M})^{-1}(c_M + \alpha A_{M,M-1}J_{M-1}) \\ &= (I - \bar{A}_M)^{-1}(\bar{c}_M + \alpha A_{M,M-1}J_{M-1}). \end{aligned}$$

If we now use this in the $m = M-1$ equation from (7), we have that:

$$\begin{aligned} J_{M-1} &= c_{M-1} + \sum_{j=M-2}^{M}\alpha A_{M-1,j}J_j \\ &= \bar{A}_{M-1}J_{M-1} + \bar{c}_{M-1} + \alpha A_{M-1,M-2}J_{M-2} \\ &= (I - \bar{A}_{M-1})^{-1}(\bar{c}_{M-1} + \alpha A_{M-1,M-2}J_{M-2}). \end{aligned}$$

By repeated substitution in the remaining equations of (7) one eventually establishes that in general:

$$J_m = (I - \bar{A}_m)^{-1}(\bar{c}_m + \alpha A_{m,m-1}J_{m-1}), \tag{12}$$

for $m = M, \ldots, 1$ and

$$J_0 = (I - \bar{A}_0)^{-1}\bar{c}_0. \tag{13}$$

The algorithm to perform a policy evaluation of R thus works as follows (pseudo code for an efficient implementation of this algorithm is given in the subsequent paragraph):

- *Step 0*: The algorithm proceeds by initialising $\bar{A}_M$, $\Theta_{k,M}$ (for $k = 0, \ldots, M-1$) and $\bar{c}_M$ according to equations (8a), (9) and (11a).

- *Step 1*: Then equations (8b), (10) and (11b) can be used to determine $\bar{A}_m$, $\Theta_{k,m}$ and $\bar{c}_m$, for $m = M-1, \ldots, 0$ and $k = 0, \ldots, m-1$ by iterating backwards.

- *Step 2*: Afterward equation (13) is used to determine J_0.

- *Step 3*: We iteratively derive J_m from J_{m-1} using equation (12) for $m = 1, \ldots, M$.

An efficient implementation:. We can reduce the memory complexity using two matrices: the $b \times (M+1)b$ matrix $V = [V_0 \ldots V_M]$ and the $b \times (M+1)$ matrix $W = [W_0 \ldots W_M]$ to store the intermediate and final results, where V_m and W_m are square matrices and column vectors of size b, respectively. The pseudo code for this implementation is given in Figure 1. At the end of step 3 W holds the long-run average cost vectors $[J_0, J_1, \ldots, J_M]$. In Step 2 we only require the $A_{k,m}$ matrices one block column at a time. As a consequence this step can be implemented in $O(b^2M)$ memory and $O(b^3M^2)$ time, which is also the overall performance of the algorithm.

The Quasi-Birth-Death (QBD) case:. If the matrices $P(a)$ are skip-free in both directions, then $A_{k,m} = 0$ if $m > k+1$ and subsequently $\Theta_{k,m}$ equals $\alpha A_{k,m}$ for $k = m-1$ and zero otherwise. Therefore, Equations (8a), (8b), (11a), (11b), (12) and (13) are equivalent to Theorem 3.1 in [10] if we let $J_m = J_{M-m}$, $\bar{A}_m = \alpha C_{M-m}$, $\bar{c}_m = g_{M-m}$ and $c_m = g_{M-m,M-m-1} + g_{M-m,M-m} + g_{M-m,M-m+1}$ (and $A_{m,m} = A_1^{(M-m)}$, $A_{m,m+1} = A_2^{(M-m)}$, $A_{m,m-1} = A_0^{(M-m)}$). The reason we need to replace m by $M-m$ is due to the fact that in [10] a left-to-right reduction was used, as is the case in the following section. For the QBD both directions result in two sets of equivalent equations, which is no longer true if the chain is only skip-free in one direction.

3.2 A policy evaluation algorithm for MDPs skip-free in one direction: the left-to-right approach

In the previous subsection we used a right-to-left approach to solve equation (7). It is also possible to work in the other direction and to start with the first equation. We define analogue matrices $\bar{A}'_m$ with $0 \le m \le M$, $\Theta'_{m,k}$ with $0 \le m < k \le M$ and vectors $\bar{c}'_m$ with $0 \le m \le M$ that are used to evaluate the long-run average cost. These matrices are recursively determined as follows:

$$\bar{A}'_0 = \alpha A_{0,0} \tag{14a}$$

$$\bar{A}'_m = \alpha(A_{m,m} + A_{m,m-1}(I - \bar{A}'_{m-1})^{-1}\Theta'_{m-1,m}), \tag{14b}$$

for $m = 1, \ldots, M$ and

$$\Theta'_{0,k} = \alpha A_{0,k}, \tag{15}$$

with $k = 1, \ldots, M$ and for $m = 1, \ldots, M$ we set

$$\Theta'_{m,k} = \alpha(A_{m,k} + A_{m,m-1}(I - \bar{A}'_{m-1})^{-1}\Theta'_{m-1,k}), \tag{16}$$

where $k = m+1, \ldots, M$. Finally we set

$$\bar{c}'_0 = c_0 \tag{17a}$$

$$\bar{c}'_m = c_m + \alpha A_{m,m-1}(I - \bar{A}'_{m-1})^{-1}\bar{c}'_{m-1}, \tag{17b}$$

for $m = 1, \ldots, M$.
We can now use these matrices and vectors to rewrite equation (7). Consider the first equation in (7), which can be rewritten as

$$\begin{aligned} J_0 &= c_0 + \alpha A_{0,0} J_0 + \alpha \sum_{i=1}^{M} A_{0,i} J_i \\ &= (I - \bar{A}'_0)^{-1}(\bar{c}'_0 + \sum_{i=1}^{M} \Theta'_{0,i} J_i). \end{aligned}$$

Using this in the second equation of (7), we find that:

$$\begin{aligned} J_1 &= c_1 + \alpha A_{1,0} J_0 + \alpha A_{1,1} J_1 + \alpha \sum_{i=2}^{M} A_{1,i} J_i \\ &= \bar{A}'_1 J_1 + \bar{c}'_1 + \sum_{i=2}^{M} \Theta'_{1,i} J_i \\ &= (I - \bar{A}'_1)^{-1}(\bar{c}'_1 + \sum_{i=2}^{M} \Theta'_{1,i} J_i). \end{aligned}$$

Repeating this argument we find the following result:

$$J_m = (I - \bar{A}'_m)^{-1}(\bar{c}'_m + \sum_{i=m+1}^{M} \Theta'_{m,i} J_i), \tag{18}$$

for $m = 0, \ldots, M-1$ and

$$J_M = (I - \bar{A}'_M)^{-1}\bar{c}'_M. \tag{19}$$

The left-to-right algorithm works as follows:

- *Step 0*: The algorithm proceeds by initialising $\bar{A}'_0$, $\Theta'_{0,k}$ (for $k = 1, \ldots, M$) and $\bar{c}'_0$ according to equations (14a), (15) and (17a).

- *Step 1*: Then equations (14b), (16) and (17b) can be used to determine $\bar{A}'_m$, $\Theta'_{m,k}$ and $\bar{c}'_m$ iteratively for $m = 1, \ldots, M$ and $k = m+1, \ldots, M$.

- *Step 2*: Afterward equation (19) is used to determine J_M.

- *Step 3*: Finally we iteratively compute J_m from J_{m+1}, $\ldots, J_M$ using equation (18).

A note on the implementation:. This algorithm is almost identical to the right-to-left algorithm, however, looking at equations (11b), (12), (17b) and (18), we see that J_m now depends on J_{m+1} to J_M, while $\bar{c}'_m$ only depends on $\bar{c}'_{m-1}$, while in the right-to-left model it was the other way around. This seems to make a low memory implementation as mentioned in the previous section problematic as there seems to be no way to avoid the need to store all the $\Theta'_{m,k}$ matrices.

The Quasi-Birth-Death (QBD) case:. If the chain P is skip-free in both directions, then $A_{m,k} = 0$ if $k > m+1$ and subsequently $\Theta'_{m,k}$ equals $\alpha A_{m,k}$ for $k = m+1$ and zero otherwise. Therefore, Equations (14a) to (17b) are equivalent to Theorem 3.1 in [10] if we let $\bar{A}'_m = \alpha C_m$, $\bar{c}'_m = g_m$ and $c_m = g_{m,m-1} + g_{m,m} + g_{m,m+1}$ (and $A_{m,m} = A_1^{(m)}$, $A_{m,m+1} = A_0^{(m)}$ and $A_{m,m-1} = A_2^{(m)}$).

- *Step 0:* Compute $\bar{A}_M, \Theta_{k,M}$ $(k = 0, \ldots, M-1)$ and $\bar{c}_M$ and set:

$$\begin{aligned} V &= [\ \Theta_{0,M} \quad \ldots \quad \Theta_{M-1,M} \quad (I - \bar{A}_M)^{-1}\ ] \\ W &= [\ \Theta_{0,M}(I - \bar{A}_M)^{-1}\bar{c}_M \quad \ldots \quad \Theta_{M-1,M}(I - \bar{A}_M)^{-1}\bar{c}_M \quad (I - \bar{A}_M)^{-1}\bar{c}_M\ ]. \end{aligned}$$

- *Step 1:* Iteratively replace V_m to V_0 and W_m to W_0, by V'_m to V'_0 and W'_m to W'_0, for $m = M-1, \ldots, 0$:

$$\begin{array}{llll} V'_m & \text{equals} & (I - \bar{A}_m)^{-1} & \text{via} \quad \bar{A}_m = \alpha(A_{m,m} + V_m V_{m+1} A_{m+1,m}) \\ V'_k & \text{equals} & \Theta_{k,m} & \text{via} \quad V'_k = \alpha(A_{k,m} + V_k V_{m+1} A_{m+1,m}) \\ W'_m & \text{equals} & (I - \bar{A}_m)^{-1}\bar{c}_m & \text{via} \quad W'_m = V'_m(c_m + W_m) \\ W'_k & \text{equals} & \sum_{j=m}^{M} \Theta_{k,j}(I - \bar{A}_j)^{-1}\bar{c}_j & \text{via} \quad W'_k = W_k + V'_k W'_m, \end{array}$$

  for $k = m-1, m-2, \ldots, 0$. End of this step:

$$\begin{aligned} V &= [\ (I - \bar{A}_0)^{-1} \quad (I - \bar{A}_1)^{-1} \quad \ldots \quad (I - \bar{A}_M)^{-1}\ ] \\ W &= [\ (I - \bar{A}_0)^{-1}\bar{c}_0 \quad (I - \bar{A}_1)^{-1}\bar{c}_1 \quad \ldots \quad (I - \bar{A}_M)^{-1}\bar{c}_M\ ]. \end{aligned}$$

- *Step 2:* $W_0 = J_0$.
- *Step 3:* For $m = 1, \ldots, M$ replace W_m by $J_m = W_m + V_m \alpha A_{m,m-1} W_{m-1}$ as defined in (12).

Figure 1: An efficient implementation for the policy evaluation algorithm: the right-to-left approach

3.3 A low memory policy iteration algorithm for MDPs skip-free in one direction

As we mentioned earlier, the right-to-left algorithm allows us to implement the policy evaluation step in $O(b^2M)$ memory and $O(b^3M^2)$ time. The policy improvement step can also be performed using $O(b^2M)$ memory by constructing $P(a)$ block row per block row for the computation of $P(a)J_{R_n}$, leading to an overall memory and time complexity of $O(b^2M)$ and $O(b^3M^2)$ per iteration. For small values of M, however, it might be possible to store the transition matrices $P(a)$ as a whole, avoiding the need to rebuild the transition matrices needed at each step, causing some reduction in the computation time.

We shall refer to the algorithm with a $O(b^2M)$ memory requirement, as the *modified* policy iteration algorithm. For a comparison in execution times of both variants see Section 5.1, where we focused on the right-to-left reduction variant only.

4. FIBRE DELAY LINES

In Section 5 we will make a comparison between the classic value iteration algorithm [9], the policy iteration algorithm by White after reblocking the system such that it becomes skip-free in both directions [10] and the new policy iteration algorithm as introduced in Section 3. First, let us briefly introduce the general notion of FDL buffers.

As in [5], we study a single outgoing wavelength in a Wavelength-division Multiplexing system and assume contention for it is resolved by means of a Fibre Delay Line (FDL) buffer, which can delay, if necessary, data packets, called optical bursts (OBs), until the channel becomes available again. Unlike conventional electronic buffers, however, it cannot delay bursts for an arbitrary period of time, but it can only realize a discrete set of N delay values. Traditionally, there are two possibilities for the delay values $a_1 \leq a_2 \leq \ldots \leq a_N$: either all delay fibres have the same length, i.e., $a_i = T$ with $i = 1, 2, \ldots, N$, or the values are equidistant, i.e., $a_i = iD$ with $i = 1, \ldots, N$, where D is termed the buffer granularity. It should be clear that a discrete set of delays may give cause to voids on the outgoing channel ([4], [2]) as one cannot always delay a packet by the appropriate amount. We do not attempt to fill these voids with other OBs as this requires a lot of intelligence and would alter the order of the OBs.

Define the scheduling horizon at time t as the earliest time $t' > t$ by which all OBs present at time t will have left the system and denote it by $\bar{H}$. When the $k-$th burst sees a scheduling horizon $\bar{H}_k$ upon arrival, with $a_i < \bar{H}_k \leq a_{i+1}$ for some $0 \leq i \leq N$ (and with $a_0 = 0$ and $a_{N+1} = \infty$), it will have to be delayed by a_{i+1} time units (if $i < N$, otherwise it is dropped), possibly creating a void on the outgoing channel (unless $\bar{H}_k = a_{i+1}$). Figure 2 shows the evolution of the scheduling horizon and the corresponding voids if $a_i = iD$ for all i.

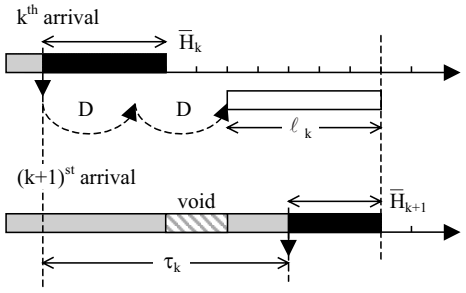

Figure 2: Evolution of the scheduling horizon $\bar{H}$ from one arrival to the next. l_k is the length of the $k-$th OB and τ_k the burst inter-arrival time

The length of the longest delay line corresponds to the maximum achievable delay a_N, therefore if an OB sees a scheduling horizon larger than a_N upon arrival, the burst is dropped (that is, lost).

4.1 Traffic scenario

The study in [5] made use of packet traces collected by the NLANR (National Laboratory for Applied Network Research). More specifically, we have used IP packet traces coming from the following two links: AIX (a measurement point that sits at the interconnection point of NASA Ames and the MAE-West interconnection of Metropolitan Fiber Systems) and COS (Colorado State University). The cumulative distributions of the packet sizes of the considered traces are depicted in Figure 3. For the comparison between the different approaches, we speed up the optimization process by clustering the packet sizes in the following way: all IP packets with a size less than or equal to 100 bytes are regarded as size 2 packets, all packets with a size between $101+(i-3)50$ and $150+(i-3)50$ bytes are considered size i (with $i = 3, \ldots, 30$) packets. Figure 4 shows the resulting packet length distributions of the clustered traces. This clustered distribution is used as the OB length distribution when comparing the different strategies. For simplicity we restrict ourselves to geometric inter-arrival times and we use an FDL buffer with $N = 10$ FDLs.

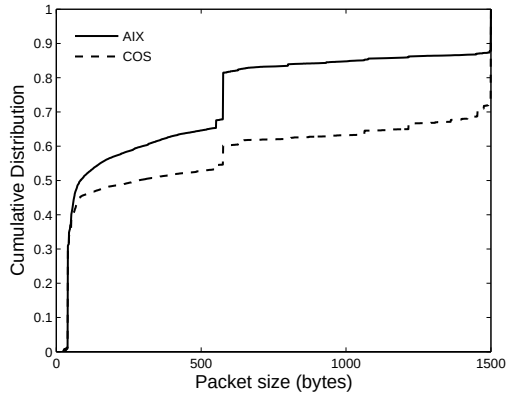

Figure 3: The complete IP packet length distribution from COS and AIX

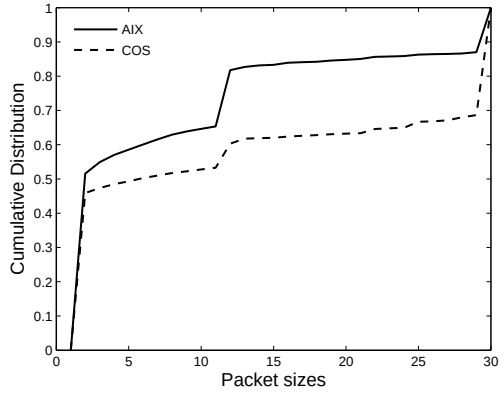

Figure 4: The clustered IP packet length distribution from COS and AIX

For the OB arrival process we limit our description to the Bernoulli process with parameter p (for a more general discussion, see [5]). We define $B_s = pP[L = s]$, for $0 < s \leq L_{\max}$ (where L denotes the OB length distribution and $L_{\max}$ denotes the maximum packet size) and $B_0 = 1 - p$ (thus $b = 1$ and $M = a_N + L_{\max} - 1$). With some abuse of notation, let $\bar{H}_n$ be the value of the scheduling horizon at time slot n as opposed to the value of the horizon as seen by the n-th arrival. Then, $(\bar{H}_n)_{n\geq 0}$ forms a discrete-time Markov chain (MC) with a finite transition matrix P:

$$P = \begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & \ldots & A_{0,M-1} & A_{0,M} \\ A_{1,0} & A_{1,1} & A_{1,2} & \ldots & A_{1,M-1} & A_{1,M} \\ 0 & A_{2,1} & A_{2,2} & \ldots & A_{2,M-1} & A_{2,M} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & A_{M,M-1} & A_{M,M} \end{bmatrix}, \quad (20)$$

where $A_{h,h'}$ is determined as follows:

$$A_{0,h'} = \begin{cases} (1-p) + pP[L=1] & h' = 0 \\ pP[L = h'+1] & 0 < h' \leq L_{\max} - 1 \\ 0 & otherwise, \end{cases}$$

and

$$A_{h,h'} = \begin{cases} 1-p & h' = h-1 \\ pP[L = h' - a_i + 1] & a_{i-1} < h \leq a_i \leq h' \leq \\ & L_{\max} + a_i - 1, 1 \leq i \leq N \\ 0 & otherwise, \end{cases}$$

for $0 < h \leq a_N$ and

$$A_{h,h'} = \begin{cases} 1 & h' = h-1 \\ 0 & otherwise, \end{cases}$$

for $a_N < h \leq M$. Hence, the transition matrix for this setup is skip-free in one direction with $b = 1$ (see (1)). A common FDL setup exists in taking equidistant delay values with the granularity parameter D equal to $L_{\max} - 1$ [2]. This will be the basic scenario during the numerical explorations in Section 5.

4.2 Preventive drop mechanism

Let us now explain how the preventive drop mechanism might improve the loss rate of an FDL buffer. In this section, we limit ourselves to the case of Bernoulli arrivals and an equidistant delay line structure (i.e., $a_i = iD$ for $i = 1, \ldots, N$). The underlying idea is that as voids on the outgoing fibre diminish the capacity of the system, it might be worthwhile to drop optical bursts that cause large voids even though there is still buffer capacity at hand. Intuitively, such a preventive drop approach seems especially useful when the system is heavily loaded as there are plenty of other bursts, possibly causing smaller voids, available that may take advantage of the remaining buffer capacity. Hence one "bad" burst, who causes a large void, might be dropped in order to accept multiple "good" bursts. To perform this analysis we made use of a Markov Decision Process. For reasons of completeness we will briefly describe the preventive drop algorithm for equidistant FDL buffers.

We define the set of actions $\mathcal{A} = \{a_1, \ldots, a_n\}$ with $0 = q(a_1) < q(a_2) < \ldots < q(a_n) = 1$. Here $q(a)$ denotes the probability that a new OB is dropped under action a given that we are in a state $h > 0$. In state $h = 0$, OBs are always accepted meaning the transitions are identical for all actions

a_i:

$$p_{0,h'}(a) = \begin{cases} (1-p) + pP[L=1] & h'=0 \\ pP[L=h'+1] & h'>0 \\ 0 & otherwise. \end{cases}$$

For $0 < h \leq ND$ we find

$$p_{h,h'}(a) = \begin{cases} (1-p) + pq(a) & h' = h-1 \\ p(1-q(a))P[L=L_D(h')] & L_D(h') \geq 1 \\ 0 & otherwise, \end{cases}$$

where $L_D(h') = h' - D\lceil h/D \rceil + 1$. For $h > ND$, all OBs are dropped, implying

$$p_{h,h'}(a) = \begin{cases} 1 & h' = h-1 \\ 0 & otherwise. \end{cases}$$

The cost function is defined such that the long-run average costs coincide with the average number of losses per slot. This is realized by setting $c_h(a)$ as

$$c_h(a) = \begin{cases} 0 & h=0 \\ pq(a) & h=1,\ldots,ND \\ p & h=ND+1,\ldots,ND+L_{max}-1. \end{cases}$$

Thus, minimizing the long-run average cost (with α sufficiently close to 1), corresponds to minimizing the OB loss rate. Various numerical experiments, not reported here, have indicated that every action $R_{\text{opt}}(h)$ part of the optimal policy R_{opt} is either action a_1 or a_n, meaning that we either prematurely drop all the bursts that observe a scheduling horizon h or none. In other words, the lowest loss rate is realized by either dropping all bursts that make use of the k-th delay line and cause a void of size v (i.e., $h = kD - v$ with $k > 0$), or by accepting all bursts of this type. This observation allows a significant reduction in the system optimization time, because it now suffices to consider just two actions per state of the MDP process.

In [5] we solved the MDP problem using the value iteration algorithm, explained in detail in [9]. Obviously, we can also use the new policy iteration algorithm as described in Section 3. Moreover, a third option is to reblock the transition matrix P such that it becomes skip-free in both directions and to apply White's algorithm [10]. A comparison between these three approaches is given in the next section.

5. COMPARISON BETWEEN THE DIFFERENT TECHNIQUES TO SOLVE AN MDP

5.1 Comparison Results

In this section we will use the traffic scenario as described in Section 4. As $D = L_{\max} - 1$ is chosen for this experiment, we can reblock the transition matrix P (20) such that it becomes skip-free in both directions as shown in Eq. (21), where $A_1^{(0)}$ is a scalar, $A_0^{(0)}$, resp. $A_2^{(1)}$, is a vector of size $1 \times D$, resp. $D \times 1$ and the other matrices $A_n^{(m)}$ are of size $D \times D$. As a result, we can make use of the policy iteration algorithm of White [10].

Figure 5 shows the execution times as a function of the number of FDLs, with a load $\rho = 0.9$. From this figure we can conclude that the policy iteration algorithms offer a significant computational reduction compared to the value iteration algorithm, especially if the number of FDLs is large. For a small number of FDLs both policy iteration algorithms are efficient, but as the number of FDLs increases the reblocking approach becomes inferior. We also see that the *modified* policy iteration algorithm is outperformed by the policy iteration algorithm that stores the $P(a)$ matrices, due to the repeated recomputation of the (block) columns of the transition matrix. Similar results are observed in Figure 6 that shows the execution times as a function of the load in case there are $N = 10$ FDLs.

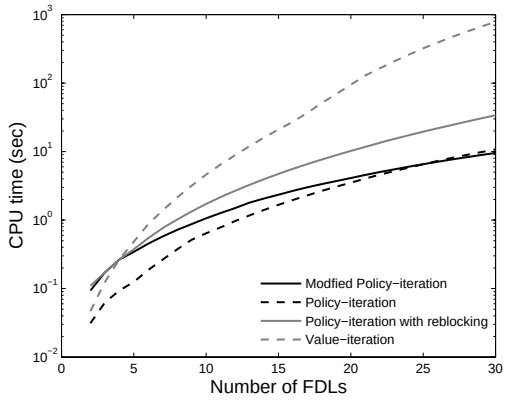

Figure 5: Execution times for the different strategies that can be used to solve an MDP as a function of the number of FDLs

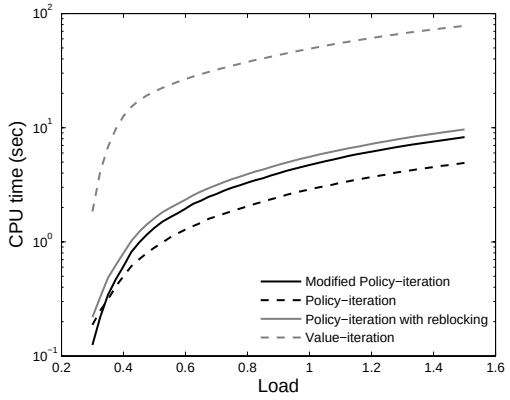

Figure 6: Execution times for the different strategies that can be used to solve an MDP as a function of the load

5.2 Results for the entire packet trace

For large values of N or $L_{\max}$ one requires a lot of memory to store the entire transition matrices $P(a)$. As described in Section 3.3 we do not need to store these matrices when using the modified policy iteration algorithm. Therefore, we can apply the preventive drop algorithm on the system using the complete AIX and COS traces instead of the clustered traces (the slots size is now 1 byte instead of 50 bytes). Although the value iteration algorithm does not require the storage of $P(0)$ or $P(1)$ as a whole, we still require clustering due to the long execution times (see Figure 5). Apart from the equidistant FDL buffer with $D = L_{\max} - 1$ and the

$$P = \begin{bmatrix} A_1^{(0)} & A_0^{(0)} & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ A_2^{(1)} & A_1^{(1)} & A_0^{(1)} & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & A_2^{(2)} & A_1^{(2)} & A_0^{(2)} & \dots & 0 & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & \dots & A_2^{(N-1)} & A_1^{(N-1)} & A_0^{(N-1)} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & A_2^{(N)} & A_1^{(N)} & A_0^{(N)} \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & A_2^{(N+1)} & A_1^{(N+1)} \end{bmatrix}. \quad (21)$$

equidistant FDL buffer with the optimal preventive drop algorithm, we also plot the results for an equidistant FDL buffer with an optimal granularity D in the range $[2, L_{\max} - 1]$. Further, we also apply the preventive drop algorithm on this optimally chosen granularity. Notice, to determine the optimal D, we cannot rely on the MDP formulation, but simply solve the system for each value of D. A detailed discussion of these results is given in [5].

Our interest goes to the comparison between the results based on the clustered traces (Figure 9 and Figure 10) and those based on the complete traces (Figure 7 and Figure 8). We observe that the clustered traces lead to some underestimation of the loss probabilities, however, the conclusions drawn from both curves with respect to the usefulness of the preventive drop mechanism are identical, which confirms our intuition that the clustered trace sufficed to get a good understanding of the system behavior.

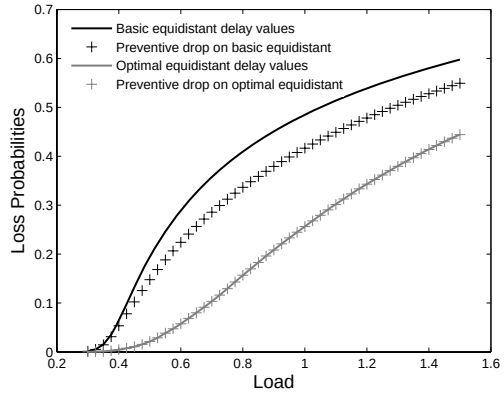

Figure 7: Comparison of the loss probabilities obtained with different methods for the complete IP packet trace (AIX)

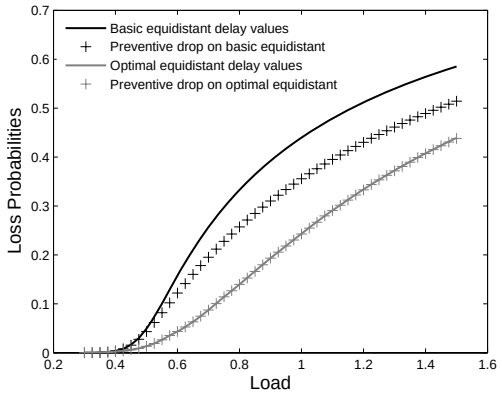

Figure 8: Comparison of the loss probabilities obtained with different methods for the complete IP packet trace (COS)

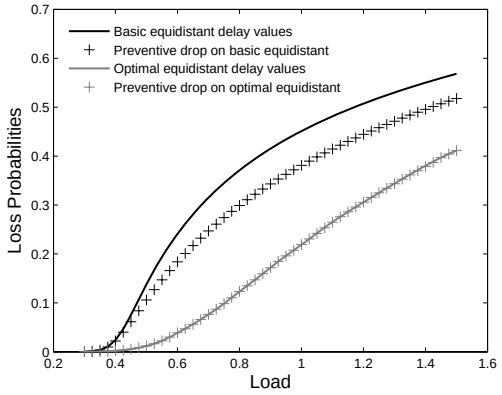

Figure 9: Comparison of the loss probabilities obtained with different methods for the clustered IP packet trace (AIX)

Acknowledgment

B. Van Houdt is a post-doctoral Fellow of the FWO-Flanders.

6. REFERENCES

[1] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd ed. edition, 2001.

[2] F. Callegati. Optical buffers for variable length packet switching. *IEEE Communications Letters*, 4:292–294, 2002.

[3] B. Van Houdt, K. Laevens, J. Lambert, C. Blondia, and H. Bruneel. Channel utilization and loss rate in a single-wavelength fibre delay line (FDL) buffer. In *Proceedings of IEEE Globecom 2004, paper OC05-07*, Dallas USA, November 2004.

[4] K. Laevens and H. Bruneel. Analysis of a single wavelength optical buffer. In *Proceedings of Infocom*, San Francisco, April 2003.

[5] J. Lambert, B. Van Houdt, and C. Blondia. Single-wavelength optical buffers: non-equidistant structures and preventive drop mechanisms. In *Proceedings of the 2005 Networking and Electronic*

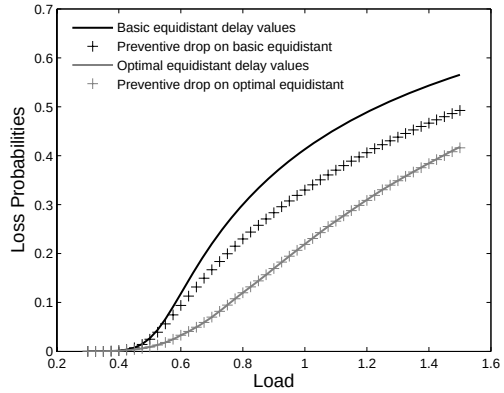

Figure 10: Comparison of the loss probabilities obtained with different methods for the clustered IP packet trace (COS)

Commerce Research Conference (NAEC 2005), pages 545–555, Riva del Garda, 2005.

[6] J. Lambert, B. Van Houdt, and C. Blondia. A preventive conversion mechanism for conflict resolution in optical burst switched networks. In *Proceedings of 10th Conference on Optical Network Design and Modelling (ONDM 2006)*, Copenhagen, 2006.

[7] G. Latouche, P.A. Jacobs, and D.P. Gaver. Finite markov chain models skip-free in one direction. *Naval Research Logistics Quarterly*, 31:571–588, 1984.

[8] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods and stochastic modeling*. SIAM, Philadelphia, 1999.

[9] H. C. Tijms. *Stochastic Modelling and Analysis, A Computational Approach*. Wiley, 1986.

[10] L.B. White. A new policy evaluation algorithm for markov decision processes with quasi birth-death structure. *Stochastic Models*, 21:785–797, 2005.