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

J. Lambert, B. Van Houdt, C. Blondia
PATS Research Group, IBBT & University of Antwerp,
Middelheimlaan 1, B-2020 Antwerp - Belgium

I. INTRODUCTION

Markov decision processes (MDP) [1] provide a mathematical framework for studying a wide range of optimization problems. 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, 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 $P(a)$, characterizing the transition probabilities given that action a is executed, are structured; policy iteration can potentially also exploit this structure to reduce the amount of computation time and memory needed during a single 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. The approach taken by White is generalized in this work by developing an algorithm for the case where the $P(a)$ matrices are skip-free in one direction only.

MDPs have a wide range of application areas. Our motivation lies in understanding the behavior of optical fibre delay line (FDL) buffers [2], [4], [3] and [5]. To improve the loss rate of the equidistant FDL buffer, we devised a new mechanism in [5], called the preventive drop mechanism, for which the optimal drop policy 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. Using the new approach, we are now in a position to tackle larger systems, while reducing the computation times. For more details on the analysis of FDL buffers, we refer to [6].

II. MDP SKIP-FREE IN ONE DIRECTION

A. *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 $x = xP$ and $xe = 1$, where e is a column vector with all entries equal to one. In this work we will restrict ourselves to the case where $b_0 = \dots = b_M = b = K/(M+1)$, meaning all subblocks are square and have the same dimension.

B. *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].

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

$$J^* = \min_a (c(a) + \alpha P(a) J^*), \quad (3)$$

Dagstuhl Seminar Proceedings 07461
Numerical Methods for Structured Markov Chains
http://drops.dagstuhl.de/opus/volltexte/2008/1403

with the vector $J^* = (J^*(1), J^*(2), \ldots, J^*(K))$, 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. For further use, we write the size K vector $c(a)$ as $(c_0(a), c_1(a), \ldots, c_M(a))^T$, with $c_i(a)$ a size b vector for $i = 0, \ldots, M$. Equation (3) is a set of non-linear equations that in general cannot be solved directly. For a given policy R, the associated long-run average cost satisfies the equation (see [1]):

$$J_R = c(a) + \alpha P(a) J_R, \tag{4}$$

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^* is discussed in detail in [1]. The most expensive step of the algorithm is the policy evaluation step where we evaluate the policy R_n by solving (4) to yield J_{R_n}. In Section III we will study this policy evaluation step.

An MDP, 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.

III. Matrix analytic techniques for policy evaluation

In this section we drop all explicit references to the policy R under evaluation to make the notations easier. 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 (4) can be written as the set of equations:

$$\begin{aligned} J_0 &= c_0 + \sum_{i=0}^{M} (\alpha A_{0,i} J_i) \\ J_m &= c_m + \sum_{i=m-1}^{M} (\alpha A_{m,i} J_i), \end{aligned} \tag{5}$$

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

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. 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 linear reduction is performed: either from right-to-left or from left-to-right.

A. Right-to-left approach for MDPs skip-free in one direction

Analogue to the linear 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 $J(h)$. Set $\bar{A}_M = \alpha A_{M,M}$ and

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

for $m = M-1, \ldots, 0$. Besides we define $\Theta_{k,M} = \alpha A_{k,M}$ with $k = 0, \ldots, M-1$ and for $m = M-1, \ldots, 1$ we set

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

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

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

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

Let us now use these matrices and vectors to rewrite equation (5) as follows:

$$J_m = (I - \bar{A}_m)^{-1} (\bar{c}_m + \alpha A_{m,m-1} J_{m-1}), \tag{6}$$

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

$$J_0 = (I - \bar{A}_0)^{-1} \bar{c}_0. \tag{7}$$

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 Figure 1):

- *Step 0*: The algorithm proceeds by initialising $\bar{A}_M$, $\Theta_{k,M}$ (for $k = 0, \ldots, M-1$) and $\bar{c}_M$.
- *Step 1*: We can 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 (7) is used to determine J_0.
- *Step 3*: We iteratively derive J_m from J_{m-1} using equation (6) for $m = 1, \ldots, M$.

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 $O(b^2 M)$ memory and $O(b^3 M^2)$ time.

The overall performance of the algorithm can be further lowered by constructing $P(a)$ block row per block row for the computation of $P(a) J_{R_n}$ in the policy improvement step, leading to an overall memory and time complexity of $O(b^2 M)$ and $O(b^3 M^2)$ 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.

We shall refer to the algorithm with a $O(b^2 M)$ memory requirement, as the *modified* policy iteration algorithm.

B. Left-to-right approach for MDPs skip-free in one direction

In the previous subsection we used a right-to-left approach to solve equation (5). It is also possible to work in the other direction and to start with the first equation. We define

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

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

- *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 $m = M-1, \ldots, 0$:

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

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

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

- *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 (6).

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

analogue matrices $\bar{A}'_0 = \alpha A_{0,0}$ and for $m = 1, \ldots, M$ we set $\bar{A}'_m = \alpha(A_{m,m} + A_{m,m-1}(I - \bar{A}'_{m-1})^{-1}\Theta'_{m-1,m})$. $\Theta'_{0,k} = \alpha A_{0,k}$ with $k = 1, \ldots, M$ and for $m = 1, \ldots, M$ we set $\Theta'_{m,k} = \alpha(A_{m,k} + A_{m,m-1}(I - \bar{A}'_{m-1})^{-1}\Theta'_{m-1,k})$ where $k = m+1, \ldots, M$. Finally we set $\bar{c}'_0 = c_0$ and $\bar{c}'_m = c_m + \alpha A_{m,m-1}(I - \bar{A}'_{m-1})^{-1}\bar{c}'_{m-1}$ for $m = 1, \ldots, M$. We can now use these matrices and vectors to rewrite equation (5):

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

for $m = 0, \ldots, M-1$ and $J_M = (I - \bar{A}'_M)^{-1}\bar{c}'_M$. The left-to-right algorithm is almost identical to the right-to-left algorithm, however, looking at the definitions of $\bar{c}_m$ and $\bar{c}'_m$ and the equations of J_m in both variants, 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.

C. MDPs skip-free in both directions

If the matrices $P(a)$ are skip-free in both directions, the right-to-left and the left-to-right approach both can be used to solve the MDP after some minor modifications. For the details, we refer to [6].

IV. Comparison between the different techniques to solve an MDP

In [6] we made a comparative study 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 right-to-left variant of the new policy iteration algorithm (see Section III). The main results of this comparison were that the policy iteration algorithms offer a significant computational reduction in comparison with the value iteration algorithm. For a small number of FDLs, respectively for a low load, both policy iteration algorithms are efficient, but as the number of FDLs, respectively the load, increases, the reblocking approach becomes inferior. We also observed in this study 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, but requires significantly less memory.

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

[6] J. Lambert, B. Van Houdt, and C. Blondia. A policy iteration algorithm for Markov decision processes skip-free in one direction. In *Proceedings of the International Workshop on Tools for solving Structured Markov Chains (SMCtools 2007)*, Nantes, 2007.

[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.