Journal of Artificial Intelligence Research 50 (2014) 763-803 Submitted 1/14; published 8/14

Policy Iteration Based on Stochastic Factorization

André M. S. Barreto AMSB@LNCC.BR
Laboratório Nacional de Computação Científica
Petrópolis, Brazil

Joelle Pineau JPINEAU@CS.MCGILL.CA
Doina Precup DPRECUP@CS.MCGILL.CA
School of Computer Science
McGill University
Montreal, Canada

Abstract

When a transition probability matrix is represented as the product of two stochastic matrices, one can swap the factors of the multiplication to obtain another transition matrix that retains some fundamental characteristics of the original. Since the derived matrix can be much smaller than its precursor, this property can be exploited to create a compact version of a Markov decision process (MDP), and hence to reduce the computational cost of dynamic programming. Building on this idea, this paper presents an approximate policy iteration algorithm called *policy iteration based on stochastic factorization*, or PISF for short. In terms of computational complexity, PISF replaces standard policy iteration's cubic dependence on the size of the MDP with a function that grows only linearly with the number of states in the model. The proposed algorithm also enjoys nice theoretical properties: it always terminates after a finite number of iterations and returns a decision policy whose performance only depends on the quality of the stochastic factorization. In particular, if the approximation error in the factorization is sufficiently small, PISF computes the optimal value function of the MDP. The paper also discusses practical ways of factoring an MDP and illustrates the usefulness of the proposed algorithm with an application involving a large-scale decision problem of real economical interest.

1. Introduction

Decisions rarely come up alone in real situations: usually, the outcome of a decision has an effect over the next one, which in turn impacts on the next, and so on. Thus, a choice that seems beneficial from a short-sighted perspective may reveal itself to be disastrous in the long run. When dealing with a succession of interrelated choices, one must weigh the immediate effects of a decision against its long-term consequences in order to achieve good overall performance. Formally, tasks involving this trade-off between short- and long-term benefits are called *sequential decision-making problems*.

This work focuses on a particular decision-making model known as a *Markov decision process* (MDP, Puterman, 1994). An MDP is a simple yet important mathematical model that describes a sequential decision task in terms of transition probabilities and rewards. The transition probabilities represent the dynamics of the process, while the rewards provide evaluative feedback for the decisions made. Given an MDP, one is usually interested in finding an *optimal decision policy*, which maximizes the expected total reward the decision maker will receive in the long run. The natural way to perform such a search is to resort to *dynamic programming*, a class of methods for solving sequential decision problems developed concomitantly with the MDP model (Bellman, 1957).

Since the publication of Bellman's (1957) seminal book, dynamic programming has been studied for more than 50 years, and is now supported by a strong and well understood theoretical basis. Besides, it has long ago transcended the limits of academia to be tested in real situations (White, 1985, 1988, 1993). Despite the success of dynamic programming in several applications, there is a serious obstacle that hinders its widespread use: the computational cost of dynamic programming algorithms grows fast with the number of states of a problem, which precludes their use in many domains. This limitation was noted by Bellman (1961), who also pointed out that the number of states of a decision process grows exponentially with the number of dimensions of its state space—a problem that came to be known as dynamic programming's *curse of dimensionality*.

Nowadays there is a consensus that, in order to solve large-scale sequential decision problems, one must exploit special structure in the corresponding model or resort to some form of approximation (Bertsekas & Tsitsiklis, 1996; Sutton & Barto, 1998; Powell, 2007). One way of incorporating approximation into the dynamic programming framework is to create a compact version of an MDP that retains as much as possible of the information contained in the original model. The approach presented in this paper is based on this idea. Specifically, it builds on the following insight: when a transition probability matrix is approximated by the product of two stochastic matrices, one can *swap* the factors of the multiplication to obtain another transition matrix, possibly much smaller than the original, which is related to its precursor. This property, called here the "*stochastic-factorization trick*," can be exploited to create a compact version of an MDP, and hence to reduce the computational demands of dynamic programming. The main contribution of the paper is an approximate policy iteration algorithm named *policy iteration based on stochastic factorization* (PISF). As will be shown, the performance of the decision policy computed by PISF only depends on the quality of the stochastic factorization; in particular, an exact factorization leads to an optimal policy. Moreover, the computational complexity of each iteration of the proposed algorithm is only linear in the number of states of the MDP.

The stochastic factorization will be presented in detail in Section 2.4. Although simple, the presentation depends on a few basic concepts, which will be introduced in Sections 2.2 and 2.3. Section 3 discusses the use of stochastic factorization to approximate an MDP. This section also introduces and analyzes the PISF algorithm, the main contribution of the paper. Section 4 investigates the computational issues surrounding the use of PISF in practice and presents possible solutions to efficiently compute the factorization of an MDP. In Section 5 some of the proposed solutions are put to the test in a large-scale decision problem involving the maintenance of an asset with components that deteriorate over time. Section 6 outlines the relationship between the stochastic factorization and other approaches described in the literature. The paper ends in Section 7, where a brief summary is presented along with suggestions for future research.

2. Background

This section introduces the notation adopted and briefly reviews some concepts that will be used throughout the paper.

2.1 Notation

Boldface letters will be used to denote matrices and vectors. Given a matrix $\mathbf{A}$, the symbol $\mathbf{a}_i$ is used to represent its i^{th} *row*; a_{ij} denotes the j^{th} element of vector $\mathbf{a}_i$. Inequalities should be interpreted element-wise; thus $\mathbf{A} \geq \mathbf{B}$ means that $a_{ij} \geq b_{ij}$ for all i and all j. The operators 'max' and 'argmax' are applied row-by-row, that is, given $\mathbf{A} \in \mathbb{R}^{p \times q}$, $\max \mathbf{A}$ is a vector $\mathbf{b} \in \mathbb{R}^p$ such that $b_i = \max_j a_{ij}$ for all i. Finally, the symbols $b_{\max}$ and $b_{\min}$ are used as a shorthand for $\max_i b_i$ and $\min_i b_i$.

2.2 Markov Decision Processes

In the sequential decision-making model considered here decisions are made at discrete time steps. At each instant t the decision maker occupies a state $s_i \in S$ and must choose an action a from a set A. The sets S and A are called the state and action spaces, respectively. In this paper it is assumed that both S and A are finite (though possibly large). The execution of action a in state s_i moves the decision maker to a new state s_j, where a new action must be selected, and so on. Each transition $s_i \xrightarrow{a} s_j$ has a certain probability of occurrence and is associated with a reward $r \in \mathbb{R}$. The goal of the decision maker is to find a policy $\pi : S \mapsto A$, that is, a mapping from states to actions, that maximizes the expected return associated with every state in S.[1] The return is defined as follows:

$$R_\gamma(s_i) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-1} r_{t+T} = \sum_{k=1}^{T} \gamma^{k-1} r_{t+k}, \tag{1}$$

where $r_{t+k} \in \mathbb{R}$ is the reward received on the k^{th} transition starting from state s_i at time step t. The parameter $\gamma \in [0,1)$ is the discount factor, which determines the relative importance of individual rewards depending on how far in the future they are received. The sequential decision process may last forever ($T = \infty$) or until the decision maker reaches a terminal state ($T < \infty$).

The decision-making process described above can be formalized as a Markov decision process, or MDP for short. An MDP is a tuple $M \equiv (S, A, P, R, \gamma)$ (Puterman, 1994). The element P is a family of transition probability functions, one for each action $a \in A$. The function $P^a : S \times S \mapsto [0,1]$ gives the transition probabilities associated with action a; $P^a(s_j|s_i)$ is the probability of a transition to state s_j when action a is executed in state s_i. Note that $\sum_{j=1}^{|S|} P^a(s_j|s_i) = 1$, for all $a \in A$ and all $s_i \in S$ (in this paper $|\cdot|$ is used to denote both the cardinality of a set and the absolute value of a scalar; the distinction should be clear by the context). The remaining component of an MDP, R, is defined analogously to P: the reward received at transition $s_i \xrightarrow{a} s_j$ is given by $R^a(s_i, s_j)$, with $\left|R^a(s_i, s_j)\right| \leq R_{\max} < \infty$. Usually one is interested in the *expected* reward resulting from the execution of action a in state s_i, that is, $r^a(s_i) = \sum_{j=1}^{|S|} R^a(s_i, s_j) P^a(s_j|s_i)$. A policy π defined over an MDP induces a *Markov process* M^π. The dynamics of M^π are given by $P^{\pi(s_i)}(\cdot|s_i)$, where $\pi(s_i)$ is the action selected by π in state s_i. Likewise, the expected reward to be collected by π in state s_i is given by $r^{\pi(s_i)}(s_i)$.

When both the state space S and the action space A are finite, the MDP can be represented in matrix form. Each function P^a becomes a matrix $\mathbf{P}^a \in \mathbb{R}^{|S| \times |S|}$, with $p^a_{ij} = P^a(s_j|s_i)$. Since the elements in each row of $\mathbf{P}^a$ are nonnegative and sum to one, this is a stochastic matrix (see Definition 1). Stochastic matrices will play an important role in the rest of the paper. In matrix

1. More generally, decision policies are rules associating states to actions, and can range in generality from randomized history-dependent to stationary deterministic (Puterman, 1994, Section 2.1.5). Since in discounted MDPs with finite state and action spaces there always exists a stationary deterministic policy that performs optimally, this paper will focus on this class of decision policies (Puterman, 1994, Thm. 6.2.7).

form, each function r^a is a vector $\mathbf{r}^a \in \mathbb{R}^{|S|}$, where $r_i^a = r^a(s_i)$. Thus, a finite MDP M can be represented by $|A|$ matrices $\mathbf{P}^a$ and the same number of vectors $\mathbf{r}^a$: $M \equiv (S, A, \mathbf{P}^a, \mathbf{r}^a, \gamma)$. A decision policy defined over a finite MDP is a vector $\pi \in A^{|S|}$ whose element π_i is the action selected by π in s_i. The Markov process induced by π can be represented by a matrix $\mathbf{P}^\pi \in \mathbb{R}^{|S|\times|S|}$ and a vector $\mathbf{r}^\pi \in \mathbb{R}^{|S|}$. The i^{th} row of $\mathbf{P}^\pi$ corresponds to the row with the same index in the matrix $\mathbf{P}^{\pi_i}$, that is, $\mathbf{p}_i^\pi = \mathbf{p}_i^{\pi_i}$. The entries of $\mathbf{r}^\pi$ are given by $r_i^\pi = r_i^{\pi_i}$.

In this paper it is assumed that A is a totally ordered set, and the symbol 'a' is used to refer to both the action a itself and its index in A. This slight abuse of notation simplifies the presentation considerably, and the distinction should be clear from the context.

2.3 Dynamic Programming

All dynamic programming theory is built upon the concept of a *value function*. The value of a state s_i under a policy π, denoted by $V^\pi(s_i)$, is the expected return the decision maker will receive from s_i when following π. Using (1), one can write $V^\pi(s_i) = E^\pi\{R_\gamma(s_i)\}$. In the case of a finite state space, the value function is a vector $\mathbf{v}^\pi \in \mathbb{R}^{|S|}$. The vector $\mathbf{v}^\pi$ makes it possible to impose a partial ordering over decision policies. In particular, a policy π' is considered to be at least as good as another policy π if $\mathbf{v}^{\pi'} \geq \mathbf{v}^\pi$. The goal of the sequential decision problem is to find an optimal policy π^* such that $\mathbf{v}^* \geq \mathbf{v}^\pi$ for all π. It is well known that there always exists at least one such policy for a given MDP (Bertsekas, 1987; Puterman, 1994). When there is more than one optimal policy, they all share the same value function $\mathbf{v}^*$.

What makes the search for an optimal policy feasible is the *Bellman equation*, a recursive relation between state values that lies at the core of all dynamic programming algorithms. The Bellman equation of a decision policy π is given by $\mathbf{v}^\pi = \mathbf{r}^\pi + \gamma\mathbf{P}^\pi\mathbf{v}^\pi$. It is possible to use this equation to compute the value function of policy π. One way to do so is to simply convert it into the so-called Bellman operator of π, $T^\pi\mathbf{v} = \mathbf{r}^\pi + \gamma\mathbf{P}^\pi\mathbf{v}$. It is known that $(T^\pi)^t\mathbf{v} \to \mathbf{v}^\pi$ as $t \to \infty$ for any vector $\mathbf{v} \in \mathbb{R}^{|S|}$ (Bertsekas, 1987; Puterman, 1994). A more direct approach to compute $\mathbf{v}^\pi$ is to interpret the Bellman equation as a system of linear equations and compute the value function as $\mathbf{v}^\pi = (\mathbf{I} - \gamma\mathbf{P}^\pi)^{-1}\mathbf{r}^\pi$, where $\mathbf{I}$ is the identity matrix of dimension $|S|$.

Given $\mathbf{v}^\pi$, it is possible to generate a decision policy whose performance is at least as good as that of the original policy π. Let $\Omega : \mathbb{R}^{|S|} \mapsto \mathbb{R}^{|S|\times|A|}$ be a mapping associated with a given MDP M such that if $\Omega\mathbf{v} = \mathbf{Q}$, then the a^{th} column of $\mathbf{Q}$ is

$$\mathbf{q}^a = \mathbf{r}^a + \gamma\mathbf{P}^a\mathbf{v}. \tag{2}$$

It should be clear that $(T^\pi\mathbf{v})_i = (\Omega\mathbf{v})_{ia}$, with $a = \pi_i$. If instead $a = \text{argmax}_j(\Omega\mathbf{v})_{ij}$, one has the Bellman operator of the MDP—that is, $T\mathbf{v} = \max\Omega\mathbf{v}$. Alternatively, $T\mathbf{v}$ can be viewed as a single application of $T^{\pi'}$, with $\pi' = \text{argmax}\,\Omega\mathbf{v}$. The driving force of dynamic programming is the fact that if π' is derived from $\mathbf{v}^\pi$ *it cannot perform worse than* π. Therefore, all dynamic programming algorithms are variations of the same basic scheme: starting from an arbitrary $\mathbf{v} \in \mathbb{R}^{|S|}$, compute policy $\pi = \text{argmax}\,\Omega\mathbf{v}$ and apply the update rule $\mathbf{v}' \leftarrow (T^\pi)^t\mathbf{v}$ for some $t > 0$. Then, based on $\mathbf{v}'$, compute a new decision policy π', apply $T^{\pi'}$ for t steps, and so on. It can be shown that, regardless of the value of t, this process will eventually converge to an optimal policy π^* (Bertsekas, 1987; Puterman, 1994).

When the scheme described above is adopted with $t = 1$, the process reduces to successive applications of the Bellman operator T, and the resulting method is the popular value iteration algorithm. This paper will focus on the other extreme of the spectrum, when $t = \infty$. In this case one has the *policy iteration* method (Howard, 1960). Algorithm 1 shows a step-by-step description of the computations performed by policy iteration (see also Appendix A.1).

Algorithm 1 Policy iteration

Require: MDP M: $\mathbf{P}^a \in \mathbb{R}^{|S|\times|S|}$ and $\mathbf{r}^a \in \mathbb{R}^{|S|}$ for each $a \in A$, $\gamma \in [0,1)$
Ensure: π^*
$\pi' \leftarrow$ random vector in $A^{|S|}$
repeat
    $\pi \leftarrow \pi'$
    for $i \leftarrow 1,2,...,|S|$ **do** $\mathbf{p}_i^{\pi} \leftarrow \mathbf{p}_i^{\pi_i}$ **and** $r_i^{\pi} \leftarrow r_i^{\pi_i}$
    $\mathbf{v}^{\pi} \leftarrow (\mathbf{I} - \gamma \mathbf{P}^{\pi})^{-1}\mathbf{r}^{\pi}$
    $\pi' \leftarrow \operatorname{argmax} \Omega \mathbf{v}^{\pi}$ ▷ Ties are broken randomly
until $\pi = \pi'$

2.4 Stochastic-Factorization Trick

This section presents the stochastic-factorization trick, a mathematical concept recently introduced by Barreto and Fragoso (2011) that will serve as a cornerstone for the subsequent developments. The trick builds on the following definitions:

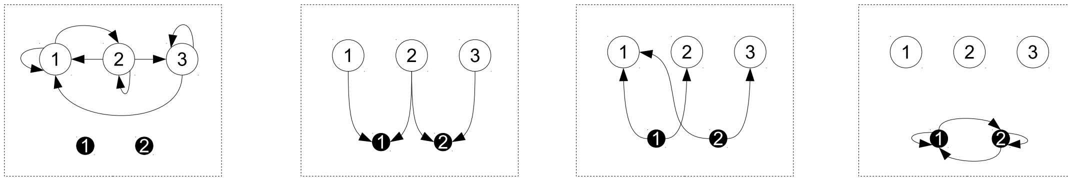
Definition 1. *A matrix* $\mathbf{P} \in \mathbb{R}^{n\times z}$ *is called* stochastic *if and only if* $p_{ij} \geq 0$ *for all* i, j *and* $\sum_{j=1}^{z} p_{ij} = 1$ *for all* i. *A square stochastic matrix is called a* transition matrix.

Definition 2. *Given a stochastic matrix* $\mathbf{P} \in \mathbb{R}^{n\times z}$, *the relation* $\mathbf{P} = \mathbf{DK}$ *is called a* stochastic factorization *of* $\mathbf{P}$ *if* $\mathbf{D} \in \mathbb{R}^{n\times m}$ *and* $\mathbf{K} \in \mathbb{R}^{m\times z}$ *are also stochastic matrices. The integer* $m > 0$ *is the* order *of the factorization.*

The relation $\mathbf{P} = \mathbf{DK}$ and the fact that $\mathbf{D}$ is stochastic imply that every row of $\mathbf{P}$ can be obtained as a convex linear combination of the rows of $\mathbf{K}$. In other words, all n stochastic vectors $\mathbf{p}_i \in \mathbb{R}^{1\times z}$ lie within the convex hull defined by the set of m stochastic vectors $\mathbf{k}_i \in \mathbb{R}^{1\times z}$. Obviously, when $m \geq n$ it is always possible to find such a hull, that is, it is always possible to compute a stochastic factorization of $\mathbf{P}$. When $m < n$, however, an exact factorization might not be possible. This leads to the following definition:

Definition 3. *The* stochastic rank *of a stochastic matrix* $\mathbf{P} \in \mathbb{R}^{n\times z}$, *denoted by* $\mathrm{srk}(\mathbf{P})$, *is the smallest possible order of the stochastic factorization* $\mathbf{P} = \mathbf{DK}$.

A matrix is called nonnegative if all its elements are greater than or equal to zero. That said, the definitions of *nonnegative factorization* and *nonnegative rank* follow analogously to that of their stochastic counterparts. Cohen and Rothblum (1991) have shown that it is always possible to derive a stochastic factorization from a nonnegative factorization of a stochastic matrix (see their Theorem 3.2). Since any stochastic factorization is also a nonnegative factorization, it follows that the nonnegative and stochastic ranks of a stochastic matrix coincide. It is easy to show that if $\mathbf{P}$ has only one nonzero element per row, the stochastic rank of this matrix coincides with its conventional rank, that is, $\mathrm{srk}(\mathbf{P}) = \mathrm{rk}(\mathbf{P})$. In the general case, however, the only thing that can be said is that $\mathrm{rk}(\mathbf{P}) \leq \mathrm{srk}(\mathbf{P}) \leq \min(n, z)$ (Cohen & Rothblum, 1991).



$$\mathbf{P} = \begin{bmatrix} 0.10 & 0.90 & 0.00 \\ 0.28 & 0.63 & 0.09 \\ 0.70 & 0.00 & 0.30 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 1.0 & 0.0 \\ 0.7 & 0.3 \\ 0.0 & 1.0 \end{bmatrix} \quad \mathbf{K} = \begin{bmatrix} 0.1 & 0.9 & 0.0 \\ 0.7 & 0.0 & 0.3 \end{bmatrix} \quad \bar{\mathbf{P}} = \begin{bmatrix} 0.73 & 0.27 \\ 0.70 & 0.30 \end{bmatrix}$$

Figure 1: Reducing the dimension of a Markov process from $n = 3$ states to $m = 2$ artificial states. The original states are represented as white circles; black circles depict artificial states. These figures have appeared before in the article by Barreto and Fragoso (2011).

The stochastic factorization has appeared before in the literature, either as defined above (Cohen & Rothblum, 1991; Ho & van Dooren, 2007) or in slightly modified versions (Cutler & Breiman, 1994; Ding et al., 2010). However, this paper will focus on a useful property of this type of factorization that has only recently been noted (Barreto & Fragoso, 2011).

Let $\mathbf{P}$ be a transition matrix of dimension n representing the dynamics of a Markov process. The stochastic factorization $\mathbf{P} = \mathbf{DK}$ admits an interesting interpretation in this case. Suppose $m < n$, where m is the order of the factorization. The elements in each row of $\mathbf{D}$ can be seen as transition probabilities from the states of the original Markov process to a set of m artificial states. Similarly, the rows of $\mathbf{K}$ may be interpreted as probabilities of transitions in the opposite direction. With this interpretation in mind, it is interesting to ask why the product $\mathbf{DK}$ restitutes the dynamics of the original process. To answer this question, it suffices to see each element $p_{ij} = \sum_{l=1}^{m} d_{il} k_{lj}$ as the sum of the probabilities associated with m two-step transitions: from s_i to each artificial state and from these back to s_j. In other words, p_{ij} is the accumulated probability of all possible paths from s_i to s_j with a stopover in one of the artificial states. Following similar reasoning, it is not difficult to see that by *swapping* the factors of the stochastic factorization, that is, by switching from $\mathbf{DK}$ to $\mathbf{KD}$, one obtains the transition probabilities between the artificial states. This makes it possible to define a new Markov process, composed of m artificial states, whose dynamics are given by $\bar{\mathbf{P}} = \mathbf{KD}$. Figure 1 illustrates this idea for the case in which a Markov process with three states is reduced to a compact model containing only two artificial states.

By simply swapping the factors of a stochastic factorization, it is possible to derive a new matrix $\bar{\mathbf{P}}$ that retains information about the dynamics of the original Markov process in a compact way. The stochasticity of $\bar{\mathbf{P}}$ follows immediately from the same property of $\mathbf{D}$ and $\mathbf{K}$. What is perhaps more surprising is the fact that this matrix shares some fundamental characteristics with the original matrix $\mathbf{P}$. Specifically, it is possible to show that: (i) for each recurrent class in $\mathbf{P}$ there is a corresponding class in $\bar{\mathbf{P}}$ with the same period and, given some simple assumptions about the factorization, (ii) $\mathbf{P}$ is irreducible if and only if $\bar{\mathbf{P}}$ is irreducible and (iii) $\mathbf{P}$ is regular if and only if $\bar{\mathbf{P}}$ is regular (see Barreto & Fragoso, 2011, for details and formal definitions). This property is called here the "*stochastic-factorization trick*":

Given a stochastic factorization of a square matrix, $\mathbf{P} = \mathbf{DK}$, swapping *the factors of the factorization yields another transition matrix* $\bar{\mathbf{P}} = \mathbf{KD}$, *potentially much smaller than the original, which retains the basic topology and properties of* $\mathbf{P}$.

3. Policy Iteration Based on Stochastic Factorization

This section introduces the main contribution of the paper, an approximate policy-iteration algorithm built upon the stochastic-factorization trick. The section starts with a description of how the stochastic-factorization trick can be used to reduce the computational cost of evaluating a Markov process, and then generalizes the idea to an MDP.

3.1 Approximating a Markov Process

Suppose that during the search for an optimal policy of a given MDP one has to determine the value function of the decision policy π. Instead of computing it directly, one can generate a compact version of the Markov process induced by π and use its value function to recover the value function of the original process. The following proposition provides the mathematical foundation for the implementation of the strategy above.

Proposition 1. *Let $M \equiv (S, A, \mathbf{P}^a, \mathbf{r}^a, \gamma)$ be an MDP, with $0 \le \gamma < 1$. Given a policy $\pi \in A^{|S|}$, let $\mathbf{P}^\pi \in \mathbb{R}^{|S|\times|S|}$ and $\mathbf{r}^\pi \in \mathbb{R}^{|S|}$ be the transition probability matrix and the expected reward vector of the Markov process M^π induced by this policy in M. Let $\mathbf{D} \in \mathbb{R}^{|S|\times m}$ be a nonnegative matrix, let $\mathbf{K} \in \mathbb{R}^{m\times|S|}$ be a stochastic matrix, and let $\bar{\mathbf{r}}$ be a vector in $\mathbb{R}^m$ such that*

$$\mathbf{DK} = \mathbf{P}^\pi \text{ and } \mathbf{D}\bar{\mathbf{r}} = \mathbf{r}^\pi. \tag{3}$$

Then,

(i) $\bar{\mathbf{P}}^\pi = \mathbf{KD}$ and $\bar{\mathbf{r}}$ define a Markov process $\bar{M}^\pi$ with m states.

Let $\bar{\mathbf{v}}^\pi \in \mathbb{R}^m$ be the value function of $\bar{M}^\pi$ computed with discount factor γ. Then,

(ii) $\mathbf{v}^\pi = \mathbf{D}\bar{\mathbf{v}}^\pi$ is the value function of M^π.

Proof. Since $\mathbf{K}$ is a stochastic matrix and $\mathbf{D}$ is nonnegative, the equality $\mathbf{DK} = \mathbf{P}^\pi$ implies that $\mathbf{D}$ is also stochastic: $1 = \sum_j p_{ij} = \sum_j \sum_l d_{il} k_{lj} = \sum_l d_{il} \sum_j k_{lj} = \sum_l d_{il}$. The fact that $\mathbf{D}$ and $\mathbf{K}$ are nonnegative implies that $\bar{\mathbf{P}}^\pi$ is nonnegative; since $\sum_j \bar{p}^\pi_{ij} = \sum_j \sum_l k_{il} d_{lj} = \sum_l k_{il} \sum_j d_{lj} = \sum_l k_{il} = 1$, $\bar{\mathbf{P}}^\pi$ is a transition matrix (see Definition 1). This proves (i). In order to prove (ii), recall that $\bar{\mathbf{v}}^\pi$, the value function of $\bar{M}^\pi$, can be written as

$$\bar{\mathbf{v}}^\pi = \bar{\mathbf{r}} + \gamma \bar{\mathbf{P}}^\pi \bar{\mathbf{v}}^\pi \tag{4}$$

(the existence and uniqueness of a solution for (4) are guaranteed by the stochastic property of $\bar{\mathbf{P}}^\pi$ and the fact that $0 \le \gamma < 1$—see, for example, Lemma 2.3.3 of Golub & Loan, 1996 or Proposition 2.6 of Bertsekas & Tsitsiklis, 1996). Multiplying both sides of (4) by $\mathbf{D}$, one has

$$\mathbf{D}\bar{\mathbf{v}}^\pi = \mathbf{D}\bar{\mathbf{r}} + \gamma \mathbf{D}\bar{\mathbf{P}}^\pi \bar{\mathbf{v}}^\pi = \mathbf{r}^\pi + \gamma \mathbf{DKD}\bar{\mathbf{v}}^\pi = \mathbf{r}^\pi + \gamma \mathbf{P}^\pi \mathbf{D}\bar{\mathbf{v}}^\pi. \tag{5}$$

Expression (5) is the Bellman equation associated with the value function of M^π; since this equation has a single fixed point, (ii) must be true (Bertsekas, 1987; Puterman, 1994). □

The computation of a decision policy's value function involves $O(|S|^3)$ arithmetic operations (Littman et al., 1995). In theory, Proposition 1 makes it possible to reduce the computational complexity of such a procedure to $O(m^3)$ (also see Appendices A.1 and A.2). Practically speaking, however, the application of Proposition 1 raises some difficulties. First, one must determine a reasonable value for m, the number of artificial states in the compact model. Obviously, one wants this value to be as small as possible, but it is not trivial to find the smallest m that allows for the application of the proposition. Even if the stochastic rank of $\mathbf{P}^\pi$ is known, it might not be possible to simultaneously satisfy both equalities in (3) with $m = \text{srk}(\mathbf{P}^\pi)$. Moreover, the computation of $\mathbf{D}$, $\mathbf{K}$ and $\bar{\mathbf{r}}$ requires a number of arithmetic operations that can easily exceed the number of operations involved in the original calculation of $\mathbf{v}^\pi$.[2] For all these reasons, one may have to resort to an approximate factorization of the Markov process.

In the approximate version of the stochastic factorization problem, one is interested in finding stochastic matrices $\mathbf{D}$ and $\mathbf{K}$ that represent $\mathbf{P}^\pi$ as well as possible, *i.e.*, that minimize a measure of the dissimilarity between $\mathbf{DK}$ and $\mathbf{P}^\pi$. In order to apply Proposition 1, one must also search for a vector $\bar{\mathbf{r}} \in \mathbb{R}^m$ that makes $\mathbf{D}\bar{\mathbf{r}}$ as similar as possible to $\mathbf{r}^\pi$. Once $\mathbf{D}$, $\mathbf{K}$, and $\bar{\mathbf{r}}$ have been determined, one can swap the factors of the stochastic factorization to define a Markov process with m artificial states. As described in Proposition 1, the value function of the resulting Markov process, $\bar{\mathbf{v}}^\pi$, can be used to restore the value function of the original model. Obviously, when $\mathbf{DK} \approx \mathbf{P}^\pi$ and $\mathbf{D}\bar{\mathbf{r}} \approx \mathbf{r}^\pi$, $\mathbf{D}\bar{\mathbf{v}}^\pi$ will be, too, only an approximation of $\mathbf{v}^\pi$. An important issue in this case is to quantify the impact that errors in the approximation of $\mathbf{P}^\pi$ and $\mathbf{r}^\pi$ might have on the computation of the value function. This section provides such an analysis for a specific dissimilarity measure. In particular, it presents an upper bound for $\| \mathbf{v}^\pi - \mathbf{D}\bar{\mathbf{v}}^\pi \|_\infty$ based on $\| \mathbf{P}^\pi - \mathbf{DK} \|_\infty$ and $\| \mathbf{r}^\pi - \mathbf{D}\bar{\mathbf{r}} \|_\infty$. Here, $\| \cdot \|_\infty$ denotes the maximum norm, which induces the following norm over the space of matrices: $\| \mathbf{A} \|_\infty = \max_i \| \mathbf{a}_i \|_1 = \max_i \sum_j |a_{ij}|$.

Proposition 2. *Let $\mathbf{P}^\pi \in \mathbb{R}^{|S|\times|S|}$ and $\mathbf{r}^\pi \in \mathbb{R}^{|S|}$ be the transition probability matrix and the expected-reward vector describing the Markov process M^π induced by decision policy π. Let $\mathbf{D} \in \mathbb{R}^{|S|\times m}$ and $\mathbf{K} \in \mathbb{R}^{m\times|S|}$ be stochastic matrices, and let $\bar{\mathbf{r}}$ be a vector in $\mathbb{R}^m$. Finally, let $\bar{M}^\pi$ be the Markov process described by $\bar{\mathbf{P}}^\pi = \mathbf{KD}$ and $\bar{\mathbf{r}}$. Then,*

$$\| \mathbf{v}^\pi - \mathbf{D}\bar{\mathbf{v}}^\pi \|_\infty \leq \alpha^\pi \equiv \frac{1}{\delta^\pi}\left(\| \mathbf{r}^\pi - \mathbf{D}\bar{\mathbf{r}} \|_\infty + \frac{\gamma}{2(1-\gamma)} \| \mathbf{P}^\pi - \mathbf{DK} \|_\infty \Delta^\pi \right), \tag{6}$$

where $\mathbf{v}^\pi$ and $\bar{\mathbf{v}}^\pi$ are the value functions of M^π and $\bar{M}^\pi$, both computed with the same $\gamma \in [0,1)$, $\delta^\pi = 1 - \gamma\left(1 - \frac{1}{2}\| \mathbf{P}^\pi - \mathbf{DK} \|_\infty\right)$, and $\Delta^\pi = \max(r^\pi_{\max}, \bar{r}_{\max}) - \min(r^\pi_{\min}, \bar{r}_{\min})$.

Proof. Let $\tilde{M}^\pi$ be the Markov process with transition matrix $\mathbf{DK}$ and reward vector $\mathbf{D}\bar{\mathbf{r}}$. From Proposition 1 it follows that the value function of $\tilde{M}^\pi$ is given by $\tilde{\mathbf{v}}^\pi = \mathbf{D}\bar{\mathbf{v}}^\pi$. Recall that a Markov process can be seen as an MDP in which $|A| = 1$. Then, applying Whitt's (1978) Theorem 6.2 (b) to M^π and $\tilde{M}^\pi$, with all mappings between the two models taken to be identities, one concludes that $\tilde{v}^\pi_i \leq v^\pi_i + \alpha^\pi$ for all i. Since the mappings between M^π and $\tilde{M}^\pi$ are the identity function, one can apply Theorem 6.2 (b) again, exchanging the roles of the two models, to obtain $v^\pi_i \leq \tilde{v}^\pi_i + \alpha^\pi$ for all i. The upper bound in (6) results from the combination of the two inequalities above. $\square$

According to Whitt (1978), it is possible to construct examples showing that (6) is a tight bound. Proposition 2 makes it clear that the approximation of $\mathbf{v}^\pi$ depends on both the characteristics of M^π

2. Vavasis (2009) has shown that the determination of the nonnegative rank of a stochastic matrix is an NP-hard problem. This implies that no polynomial-time algorithm for computing $\mathbf{D}$ and $\mathbf{K}$ is currently known; if this were the case, one could compute one factorization for each value of $m = |S|, |S|-1, ..., 1$, stopping when an exact factorization is no longer possible, thus determining the matrix's rank in polynomial time.

and the quality of the stochastic factorization. First, the right-hand side of (6) increases as $\gamma \to 1$ and $\Delta^\pi \to \infty$. This is expected, since both the magnitude of the individual rewards and the rate at which they accumulate over time tend to increase the states' values. More important, the difference between $\mathbf{v}^\pi$ and $\mathbf{D}\bar{\mathbf{v}}^\pi$ directly depends on the stochastic factorization, and as $\mathbf{DK} \to \mathbf{P}^\pi$ and $\mathbf{D}\bar{\mathbf{r}} \to \mathbf{r}^\pi$ the approximation $\mathbf{D}\bar{\mathbf{v}}^\pi$ gets closer to the real value function $\mathbf{v}^\pi$. In the limit, when $\| \mathbf{r}^\pi - \mathbf{D}\bar{\mathbf{r}} \|_\infty = 0$ and $\| \mathbf{P}^\pi - \mathbf{DK} \|_\infty = 0$, one recovers Proposition 1.

It is interesting to point out an intriguing property of Proposition 2. Observe that an increase in the approximation error $\| \mathbf{P}^\pi - \mathbf{DK} \|_\infty$ has two opposite effects on the variables involved in the derived bound: if on the one hand it increases the coefficient multiplying Δ^π, as expected, on the other hand it also *decreases* the factor $1/\delta^\pi$ scaling the entire right-hand of (6). This is precisely what makes the bound tight, since the latter effect tends to alleviate the first. Needless to say, the bound is still a monotonically increasing function of $\| \mathbf{P}^\pi - \mathbf{DK} \|_\infty$, as one can easily verify by computing the appropriate partial derivative.

3.2 Approximating a Markov Decision Process

The most straightforward way to use the stochastic-factorization trick in the search for a decision policy is to factor one by one the Markov processes that come up in the search. To be more specific, let π be an arbitrary decision policy defined over a finite MDP M and let $\mathbf{P}^\pi$ and $\mathbf{r}^\pi$ describe the Markov process induced by this policy. Given approximations $\mathbf{DK} \approx \mathbf{P}^\pi$ and $\mathbf{D}\bar{\mathbf{r}} \approx \mathbf{r}^\pi$, one can define a new Markov process with transition matrix $\bar{\mathbf{P}}^\pi = \mathbf{KD}$ and reward vector $\bar{\mathbf{r}}$. This auxiliary model potentially has many fewer states than the original Markov process. After determining the value function of the compact model, $\bar{\mathbf{v}}^\pi$, one can compute an approximation of the original value function by making $\tilde{\mathbf{v}}^\pi = \mathbf{D}\bar{\mathbf{v}}^\pi$. Finally, a new policy $\pi' = \operatorname{argmax} \Omega \tilde{\mathbf{v}}^\pi$ can be derived, restarting the usual dynamic programming loop. Notice that if $\mathbf{DK} = \mathbf{P}^\pi$ and $\mathbf{D}\bar{\mathbf{r}} = \mathbf{r}^\pi$ for every policy π that comes up during the search, this process will eventually converge to an optimal decision policy. This is a direct consequence of Proposition 1.

Although feasible, the above strategy presents an obvious drawback: a new factorization must be computed for each Markov process encountered in the search for a decision policy. Another possibility is to factor the *entire* MDP at once. In this case, a possible approach is to find approximate factorizations for each Markov process M^a, $\mathbf{D}^a\mathbf{K}^a \approx \mathbf{P}^a$ and $\mathbf{D}^a\bar{\mathbf{r}}^a \approx \mathbf{r}^a$, with $a \in A$, and solve the reduced MDP $\bar{M}$ whose transition matrices are $\mathbf{K}^a\mathbf{D}^a$ and whose expected reward vectors are $\bar{\mathbf{r}}^a$. However, when the MDP $\bar{M}$ is directly solved, the quality of the solution found does not depend on the stochastic factorization only, and even an exact factorization of the MDP can lead to suboptimal decision policies (for example, as shown in Barreto, Precup, & Pineau, 2011, Prop. 1, in the particular case in which a single matrix $\mathbf{D}$ is used to factor all the Markov processes, the approximation error also depends on the "level of stochasticity" of $\mathbf{D}$, measured by $\max_i (1 - \max_j d_{ij})$). The remaining of this section discusses an alternative way of factoring an MDP in which the performance of the final decision policy depends exclusively on the quality of the stochastic factorization.

Let $M \equiv (S, A, \mathbf{P}^a, \mathbf{r}^a, \gamma)$ be a finite MDP. Let $\mathbf{D}^a \in \mathbb{R}^{|S| \times m}$ and $\mathbf{K} \in \mathbb{R}^{m \times |S|}$ be stochastic matrices such that $\mathbf{D}^a\mathbf{K} \approx \mathbf{P}^a$ for all $a \in A$, and let $\bar{\mathbf{r}} \in \mathbb{R}^m$ be a vector such that $\mathbf{D}^a\bar{\mathbf{r}} \approx \mathbf{r}^a$, again with $a \in A$. Let $\pi \in A^{|S|}$ be a policy defined on M and let M^π be the corresponding Markov process. As discussed in Section 2.3, the i^{th} row of $\mathbf{P}^\pi \in \mathbb{R}^{|S| \times |S|}$, the transition matrix of M^π, is the i^{th} row of $\mathbf{P}^{\pi_i}$, where π_i is the action selected by π in s_i (see line 4 of Algorithm 1). From the approximations $\mathbf{D}^a\mathbf{K} \approx \mathbf{P}^a$, one can see that the i^{th} row of $\mathbf{P}^{\pi_i}$ can be approximated as $\mathbf{d}_i^{\pi_i}\mathbf{K}$ (recall that $\mathbf{d}_i^{\pi_i}$ is the i^{th} row of $\mathbf{D}^{\pi_i}$).

Thus, in order to build an approximation of $\mathbf{P}^\pi$, it suffices to construct matrix $\mathbf{D}^\pi \in \mathbb{R}^{|S|\times m}$ whose rows are given by $\mathbf{d}_i^\pi = \mathbf{d}_i^{\pi_i}$. Once $\mathbf{D}^\pi$ has been determined, the transition matrix associated with π can be approximated as $\mathbf{P}^\pi \approx \mathbf{D}^\pi\mathbf{K}$. Analogously, the vector $\mathbf{r}^\pi \in \mathbb{R}^{|S|}$ can be approximated as $\mathbf{r}^\pi \approx \mathbf{D}^\pi\bar{\mathbf{r}}$.

Using the strategy above, it is straightforward to extend the ideas of Proposition 1 to a factorization of the entire MDP. Given a decision policy π, one must first compute the matrix $\mathbf{D}^\pi$ as described and then define a reduced Markov process $\bar{M}^\pi$ with transition matrix $\bar{\mathbf{P}}^\pi = \mathbf{K}\mathbf{D}^\pi$ and reward vector $\bar{\mathbf{r}}$. The corresponding value function $\bar{\mathbf{v}}^\pi$ can then be used to compute an approximation of the value function of π, which will serve as a reference for the derivation of a new decision policy π', and so on. Algorithm 2 shows how these ideas can be embedded into policy iteration, giving rise to the *policy iteration based on stochastic factorization* algorithm, or simply PISF.

Algorithm 2 Policy iteration based on stochastic factorization (PISF)

Require: $\mathbf{D}^a \in \mathbb{R}^{|S|\times m}$ for all $a \in A$, $\mathbf{K} \in \mathbb{R}^{m\times|S|}$, $\bar{\mathbf{r}} \in \mathbb{R}^m$, and $\gamma \in [0,1)$
Ensure: $\pi \approx \pi^*$
1: $\pi' \leftarrow$ random vector in $A^{|S|}$
2: **repeat**
3: $\quad \pi \leftarrow \pi'$
4: $\quad$ **for** $i \leftarrow 1,2,...,|S|$ **do** $\mathbf{d}_i^\pi \leftarrow \mathbf{d}_i^{\pi_i}$
5: $\quad \bar{\mathbf{P}}^\pi \leftarrow \mathbf{K}\mathbf{D}^\pi$
6: $\quad \bar{\mathbf{v}}^\pi \leftarrow (\mathbf{I} - \gamma\bar{\mathbf{P}}^\pi)^{-1}\bar{\mathbf{r}}$
7: $\quad$ Let $\tilde{\mathbf{Q}}^\pi \in \mathbb{R}^{|S|\times|A|}$
8: $\quad$ **for** $a \leftarrow 1,2,...,|A|$ **do** $\tilde{\mathbf{q}}^{\pi,a} \leftarrow \mathbf{D}^a\bar{\mathbf{v}}^\pi$ $\qquad \triangleright$ $\tilde{\mathbf{q}}^{\pi,a}$ is the a^{th} column of $\tilde{\mathbf{Q}}^\pi$
9: $\quad \pi' \leftarrow \operatorname{argmax} \tilde{\mathbf{Q}}^\pi$ $\qquad \triangleright$ Ties are broken randomly
10: **until** $\pi = \pi'$

In the standard policy iteration algorithm, the computation of a decision policy's value function takes $O(|S|^3)$ arithmetic operations, while the derivation of a new policy is $O(|S|^2|A|)$ (Littman et al., 1995). In contrast, PISF only needs $O(|S|m|A|)$ operations to derive a new policy—as shown in lines 8 and 9 of Algorithm 2—and breaks the value function computation into several steps whose overall complexity is $O(|S|m^2)$. The process of computing $\bar{\mathbf{v}}^\pi$ is as follows. First, one has to determine matrix $\mathbf{D}^\pi$, which involves $O(|S|)$ operations (line 4 of Algorithm 2). Then, it is necessary to perform $O(m^2|S|)$ operations to compute the transition matrix $\bar{\mathbf{P}}^\pi$ (line 5). Finally, one must calculate $\bar{\mathbf{v}}^\pi$, which is $O(m^3)$ (line 6). As one can see, the computational complexity of one iteration of PISF is *only linear in the number of states* $|S|$. Thus, when $m \ll |S|$, the number of arithmetic operations performed per iteration by PISF is much smaller than the number of operations that would be executed by the conventional policy iteration algorithm.

Regarding the space complexity of PISF, storing $\mathbf{D}^a$, $\mathbf{K}$ and $\bar{\mathbf{r}}$ requires $O(|S||A|m)$ bits. Note though that in an actual implementation the matrices $\mathbf{D}^a$ do not need to be stored in the main memory. Thus, if one is willing to trade space for time—since in this case $\mathbf{D}^\pi$ would have to be computed or loaded on demand—, the actual memory usage of the algorithm drops to $O(|S|m)$ only.

3.3 Convergence and Error Bound

The PISF algorithm rests on the approximations $\mathbf{D}^a\mathbf{K} \approx \mathbf{P}^a$ and $\mathbf{D}^a\bar{\mathbf{r}} \approx \mathbf{r}^a$. If these approximations happen to be exact, Proposition 1 is directly applicable to every decision policy generated by this algorithm, which implies that it will converge to an optimal policy π^*. Nevertheless, in order for PISF to be considered a stable method, it is necessary to show that errors in the above approximations will not cause the algorithm to behave in a completely unpredictable way. The following proposition shows that PISF is a well-behaved algorithm, in the sense that it always terminates in a finite number of iterations and the quality of the decision policy returned improves when $\|\mathbf{D}^a\mathbf{K}-\mathbf{P}^a\|_\infty \to 0$ and $\|\mathbf{D}^a\bar{\mathbf{r}}-\mathbf{r}^a\|_\infty \to 0$ for all $a \in A$.

Proposition 3. *Let $M \equiv (S, A, \mathbf{P}^a, \mathbf{r}^a, \gamma)$ be an MDP with $\gamma \in [0, 1)$. Let $\mathbf{D}^a \in \mathbb{R}^{|S|\times m}$ and $\mathbf{K} \in \mathbb{R}^{m\times|S|}$ be stochastic matrices, with $a \in A$, and let $\bar{\mathbf{r}} \in \mathbb{R}^{|S|}$. Then, if PISF is executed with $\mathbf{D}^a$, $\mathbf{K}$, $\bar{\mathbf{r}}$, and γ,*

(i) It will terminate after a finite number of iterations.

Let $\tilde{\mathbf{v}}$ be the value function of the policy returned by PISF and let $\tilde{\mathbf{r}}^a = \mathbf{D}^a\bar{\mathbf{r}}$ for all $a \in A$. Then,

(ii)
$$\| \mathbf{v}^* - \tilde{\mathbf{v}} \|_\infty \leq \frac{2}{1-\gamma}\left(\max_a \| \mathbf{r}^a - \tilde{\mathbf{r}}^a \|_\infty + \frac{\gamma}{2(1-\gamma)} \max_a \| \mathbf{P}^a - \mathbf{D}^a\mathbf{K} \|_\infty \Delta \right), \tag{7}$$

where $\mathbf{v}^*$ is the optimal value function of M and $\Delta = \max_a \tilde{r}^a_{\max} - \min_a \tilde{r}^a_{\min}$.

Proof. Let $\tilde{M} \equiv (S, A, \mathbf{D}^a\mathbf{K}, \mathbf{D}^a\bar{\mathbf{r}}, \gamma)$, that is, $\tilde{M}$ is the MDP whose transition matrices are $\mathbf{D}^a\mathbf{K}$ and whose expected reward vectors are $\mathbf{D}^a\bar{\mathbf{r}}$, for all $a \in A$. The strategy of the proof will be to show that executing PISF with $\mathbf{D}^a$, $\mathbf{K}$, $\bar{\mathbf{r}}$, and γ is equivalent to running standard policy iteration in $\tilde{M}$.

Let π, $\bar{\mathbf{v}}^\pi$ and $\tilde{\mathbf{Q}}^\pi$ be the policy, value function, and matrix computed by PISF at the i^{th} iteration (lines 3, 6, and 8 of Algorithm 2, respectively). From the definition of $\tilde{\mathbf{Q}}^\pi$, one can write

$$\tilde{\mathbf{q}}^{\pi,a} = \mathbf{D}^a\bar{\mathbf{v}}^\pi = \mathbf{D}^a(\bar{\mathbf{r}} + \gamma\bar{\mathbf{P}}^\pi\bar{\mathbf{v}}^\pi) = \mathbf{D}^a(\bar{\mathbf{r}} + \gamma\mathbf{K}\mathbf{D}^\pi\bar{\mathbf{v}}^\pi) = \mathbf{D}^a\bar{\mathbf{r}} + \gamma\mathbf{D}^a\mathbf{K}\mathbf{D}^\pi\bar{\mathbf{v}}^\pi, \tag{8}$$

where $\mathbf{D}^\pi$ is also a matrix constructed by PISF at the i^{th} iteration (line 4 of Algorithm 2). Comparing (2) and (8), it is clear that $\tilde{\mathbf{Q}}^\pi = \tilde{\Omega}\mathbf{D}^\pi\bar{\mathbf{v}}^\pi$. From Proposition 1, it follows that $\mathbf{D}^\pi\bar{\mathbf{v}}^\pi$ is the value function of the Markov process described by $\mathbf{D}^\pi\mathbf{K}$ and $\mathbf{D}^\pi\bar{\mathbf{r}}$. But this is exactly the Markov process induced by π in $\tilde{M}$ (see line 4 of Algorithm 1). Therefore, the policy computed in one iteration of PISF starting with π, $\pi' = \operatorname{argmax} \tilde{\mathbf{Q}}^\pi = \operatorname{argmax} \tilde{\Omega}\mathbf{D}^\pi\bar{\mathbf{v}}^\pi$, is the same policy that would be computed in one iteration of standard policy iteration applied to $\tilde{M}$ and also starting with π. This implies that PISF will converge to the optimal policy of $\tilde{M}$ in a finite number of iterations, and hence (i) holds (see, for example, Puterman, 1994, Thm. 6.4.2).

In order to show (ii), it suffices to resort to Whitt's (1978) results comparing "dynamic programs," a generalization of MDPs proposed by Denardo (1967). Specifically, if one applies the corollary of Whitt's Lemma 3.1 to M and $\tilde{M}$, with all mappings between the two MDPs taken to be identities, it follows that

$$\| \mathbf{v}^* - \tilde{\mathbf{v}} \|_\infty \leq \frac{2}{1-\gamma} \max_{ij} |h_{ij}|, \tag{9}$$

where h_{ij} are the elements of matrix $\mathbf{H} = \Omega\tilde{\mathbf{v}}^* - \tilde{\Omega}\tilde{\mathbf{v}}^*$ and $\tilde{\mathbf{v}}^*$ is the optimal value function of $\tilde{M}$ (since the policy computed by PISF is optimal in $\tilde{M}$, the last term appearing in Whitt's bound vanishes).

Based on Corollary (b) of Whitt's Theorem 6.1, one can write:

$$\max_{ij} |h_{ij}| \leq \max_a \| \mathbf{r}^a - \tilde{\mathbf{r}}^a \|_\infty + \frac{\gamma}{2(1-\gamma)} \max_a \| \mathbf{P}^a - \mathbf{D}^a\mathbf{K} \|_\infty \Delta. \tag{10}$$

Substituting the right-hand side of (10) in (9) one gets the desired bound.[3] □

Proposition 3 states that PISF will converge to a decision policy after a finite number of iterations. In fact, as the proof of the proposition shows, the solution returned by PISF is one of the optimal policies of the MDP $\tilde{M} \equiv (S, A, \mathbf{D}^a\mathbf{K}, \mathbf{D}^a\bar{\mathbf{r}}, \gamma)$. Thus, one can look at PISF as a fast specialized method to solve MDPs that have a specific type of structure—namely, MDPs that allow for an exact factorization with a single $\mathbf{K}$ and a single $\bar{\mathbf{r}}$. Of course, PISF will converge to a decision policy even if the factorization is not exact; in this case the algorithm becomes an approximate policy iteration method.

The error bound provided in Proposition 3 is not tight in general. This can be seen by noting that when $|A| = 1$ the bound in (7) may be looser than its counterpart in (6). Also, the bound is too pessimistic to be of practical value in many situations. Even so, Proposition 3 is of conceptual importance because it establishes the soundness of PISF. In particular, it states that the performance of the policy returned by this algorithm gets closer to optimal as the error in the MDP approximation decreases. In fact, it is not difficult to show that, if the errors in the approximations $\mathbf{D}^a\mathbf{K} \approx \mathbf{P}^a$ and $\mathbf{D}^a\bar{\mathbf{r}} \approx \mathbf{r}^a$ are below a certain threshold, the policy returned by PISF performs optimally in M. In order to do that, first note that $\tilde{\mathbf{r}}^a = \mathbf{D}^a\bar{\mathbf{r}}$ implies that $\bar{r}_{\min} \leq \tilde{r}_i^a \leq \bar{r}_{\max}$ for all a, and thus one can restrict the elements of $\bar{\mathbf{r}}$ to the interval $[\min_a r_{\min}^a, \max_a r_{\max}^a]$ and still get an exact factorization. Assuming this is the case, the term Δ appearing in the right-hand side of (7) can be replaced by $\max_a r_{\max}^a - \min_a r_{\min}^a$. This means that $\max_a \| \mathbf{P}^a - \mathbf{D}^a\mathbf{K} \|_\infty$ and $\max_a \| \mathbf{r}^a - \mathbf{D}^a\bar{\mathbf{r}} \|_\infty$ are the only terms in (7) that vary with $\mathbf{D}^a$, $\mathbf{K}$ and $\bar{\mathbf{r}}$, and hence one can make the bound arbitrarily small by driving the approximation errors to zero. Let $\Pi \subset A^{|S|}$ be the set of non-optimal policies of M and let $\varepsilon = \min_{\pi \in \Pi} \| \mathbf{v}^* - \mathbf{v}^\pi \|_\infty$. Since $\tilde{\mathbf{v}}$ is the value function of a specific policy, it follows that, if the right-hand side of (7) is smaller than ε, then $\tilde{\mathbf{v}} = \mathbf{v}^*$. Therefore, there exists a scalar ε' such that if $\max_a \| \mathbf{P}^a - \mathbf{D}^a\mathbf{K} \|_\infty < \varepsilon'$ and $\max_a \| \mathbf{r}^a - \mathbf{D}^a\bar{\mathbf{r}} \|_\infty < \varepsilon'$ the policy returned by PISF is optimal.

4. Computing the Stochastic Factorization

As shown in the previous section, PISF's performance depends crucially on the approximations $\mathbf{D}^a\mathbf{K} \approx \mathbf{P}^a$ and $\mathbf{D}^a\bar{\mathbf{r}} \approx \mathbf{r}^a$, with $a \in A$. This section discusses how to compute $\mathbf{D}^a$, $\mathbf{K}$, and $\bar{\mathbf{r}}$. It starts with a generic presentation of the problem and then gradually focuses on more specific formulations that can be solved much more efficiently.

4.1 The Optimization Problem

In order to facilitate the exposition, it will be assumed that the vectors $\mathbf{r}^a$ and matrices $\mathbf{P}^a$ have been concatenated and stacked to obtain a single matrix $\mathbf{M} \in \mathbb{R}^{|S||A| \times |S|+1}$ representing an entire MDP, as

3. Ravindran and Barto (2004) follow similar strategy to bound the approximation loss resulting from an approximate homomorphism.

follows:

$$\mathbf{M} = \begin{bmatrix} r_1^1 & p_{11}^1 & p_{12}^1 & \cdots & p_{1|S|}^1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{|S|}^1 & p_{|S|1}^1 & p_{|S|2}^1 & \cdots & p_{|S||S|}^1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_1^{|A|} & p_{11}^{|A|} & p_{12}^{|A|} & \cdots & p_{1|S|}^{|A|} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{|S|}^{|A|} & p_{|S|1}^{|A|} & p_{|S|2}^{|A|} & \cdots & p_{|S||S|}^{|A|} \end{bmatrix}. \tag{11}$$

With the representation above, the objective of the factorization problem reduces to finding matrices $\mathbf{D}$ and $\mathbf{W}$ such that $\mathbf{DW} \approx \mathbf{M}$. More formally, the approximate factorization of an MDP $\mathbf{M}$ can be formulated as a constrained nonlinear optimization problem in the following way:

Problem 1. *Given a matrix* $\mathbf{M} \in \mathbb{R}^{|S||A| \times |S|+1}$ *representing an MDP, find* $\mathbf{D} \in \mathbb{R}^{|S||A| \times m}$ *and* $\mathbf{W} \in \mathbb{R}^{m \times |S|+1}$ *in order to minimize a dissimilarity measure* $\Phi(\mathbf{M}, \mathbf{DW})$*, subjected to the following constraints:*

$$d_{ij} \geq 0 \textit{ for } i = 1, 2, \ldots |S||A| \textit{ and } j = 1, 2, \ldots, m, \tag{12}$$

$$w_{ij} \geq 0 \textit{ for } i = 1, 2, \ldots m \textit{ and } j = 2, 3, \ldots, |S|+1, \tag{13}$$

$$\textstyle\sum_{j=1}^{m} d_{ij} = 1 \textit{ for } i = 1, 2, \ldots |S||A|, \tag{14}$$

$$\textstyle\sum_{j=2}^{|S|+1} w_{ij} = 1 \textit{ for } i = 1, 2, \ldots m. \tag{15}$$

Note that the above problem formulation assumes that the first column of $\mathbf{M}$ is reserved for the rewards, as in (11). This is why the elements in the first column of $\mathbf{W}$ are not subjected to the stochasticity constraints (13) and (15). When $|A| = 1$ the model $\mathbf{M}$ is a Markov process. Also, the problem statement can be easily modified to reflect the case in which $\mathbf{M}$ represents a Markov chain (that is, when there are no rewards in the Markov process—see Barreto & Fragoso, 2011). This modification does not have any major impact on the discussion to follow.

It is not hard to see that the feasible region defined by the constraints of the above optimization problem is a convex set. Thus, if Φ is continuously differentiable, one can resort to one of the several methods presented by Bertsekas (1999) to solve a constrained nonlinear optimization problem with a convex feasible region. Among them, the most obvious choice is probably a gradient projection method, in which a candidate solution is iteratively refined by an update rule that keeps it within the problem's feasible region. Another approach that seems promising in this context is the "block coordinate descent" method (Bertsekas, 1999). In this case, only a subset of the variables is updated at a time while the remaining are kept fixed. For the optimization problem at hand, there are two blocks of variables corresponding to the elements of matrices $\mathbf{D}$ and $\mathbf{W}$. Thus, at each iteration of the algorithm one applies the following update rules:

$$\mathbf{D}' \leftarrow \underset{\mathbf{D} \in \mathbb{D}}{\operatorname{argmin}}\, \Phi(\mathbf{M}, \mathbf{DW}) \quad \text{and} \quad \mathbf{W}' \leftarrow \underset{\mathbf{W} \in \mathbb{W}}{\operatorname{argmin}}\, \Phi(\mathbf{M}, \mathbf{D}'\mathbf{W}), \tag{16}$$

where $\mathbb{D} \subset \mathbb{R}^{|S||A| \times m}$ and $\mathbb{W} \subset \mathbb{R}^{m \times |S|+1}$ are the feasible regions of $\mathbf{D}$ and $\mathbf{W}$, respectively. Depending on the characteristics of the dissimilarity measure adopted, the repeated application of the above update rules will eventually converge to a stationary point of Φ (Grippo & Sciandrone, 2000). This is true, for example, when $\Phi(\mathbf{M},\mathbf{DW}) = \| \mathbf{M}-\mathbf{DW} \|_{\mathrm{F}}$, where $\| \cdot \|_{\mathrm{F}}$ is the Frobenius norm (Cutler & Breiman, 1994; Lin, 2007b). When $\| \cdot \|_{\mathrm{F}}$ is used, one can compute $\mathbf{D}'$ and $\mathbf{W}'$ through a constrained least-squares algorithm (Cutler, 1993).

The solution of the two subproblems in (16) may require a large number of arithmetic operations. Therefore, instead of searching for exact minima, one can take a small step per iteration towards the solution of each subproblem, much like in conventional iterative descent methods (Lee & Seung, 2000). In this case, it is relatively simple to enforce the stochasticity constraints by projecting the partial solutions onto the feasible region or by incorporating a penalty term into the dissimilarity measure Φ (Lee & Seung, 1997). It should be noted, however, that when an iterative update rule is used in place of (16), the guarantee of convergence to a solution may be lost (Lin, 2007a).

The ideas above have been extensively exploited in the study of a related optimization problem known as *nonnegative matrix factorization*. As the name suggests, in this version of the problem only constraints (12) and (13) are normally imposed (Paatero & Tapper, 1994; Lee & Seung, 1999). There are many works available discussing theoretical and practical aspects of nonnegative matrix factorization (for a survey, see Berry et al., 2007). Most of the ideas discussed in these works also apply to the problem considered here. In particular, since one can derive a stochastic factorization from a nonnegative factorization of a stochastic matrix, any algorithm designed for the latter can also be used to compute the former (Cohen & Rothblum, 1991).

Instead of addressing Problem 1 as a generic nonnegative factorization, one can exploit the particular structure of matrices $\mathbf{D}$ and $\mathbf{W}$. As discussed in Section 2.4, the fact that $\mathbf{D}$ is stochastic implies that an exact factorization $\mathbf{DW} = \mathbf{M}$ is only possible when the rows of $\mathbf{M}$ are inside the convex hull defined by the rows of $\mathbf{W}$. So, one can try to solve the problem by (approximately) computing a convex hull that contains the rows of $\mathbf{M}$ and then determining the coefficients that recover $\mathbf{m}_i$ as a convex combination of the vertices of the hull—which will naturally give rise to a stochastic $\mathbf{D}$. This approach is closely related to Cutler and Breiman's (1994) "archetypal analysis" and Ding et al.'s (2010) "convex nonnegative factorization." There is also an interesting connection with the problem known as "spectral unmixing" in the field of imaging spectroscopy, in which the determination of matrix $\mathbf{W}$ is usually referred to as the task of "extracting end-members" (Keshava & Mustard, 2002; Keshava, 2003).

Yet another way of approaching Problem 1 is to impose additional constraints and resort to specialized methods. Consider for example the case in which $\mathbf{D}$ has only one nonzero element per row. In this case, the factorization can be seen as a specific instance of the well-known data-clustering problem: the vectors $\mathbf{m}_i$ must be grouped into m clusters C_j whose "centers" are the vectors $\mathbf{w}_j$—the most common choice to define the centers is to have $\mathbf{w}_j = 1/|C_j| \sum_{\{i|\mathbf{m}_i \in C_j\}} \mathbf{m}_i = 1/|C_j| \sum_{\{i|d_{ij}=1\}} \mathbf{m}_i$ (Hartigan, 1975). The goal of the clustering problem is to determine an assignment of vectors $\mathbf{m}_i$ to clusters C_j in order to minimize $\Phi(\mathbf{M},\mathbf{DW})$. Since $\mathbf{W}$ is automatically defined by a given assignment, the problem reduces to computing matrix $\mathbf{D}$. An advantage of interpreting the factorization as a clustering problem is the availability of a large number of algorithms specifically designed to solve this type of optimization (Kaufman & Rousseeuw, 1990; Gan et al., 2007). Conveniently, depending on how Φ is defined, the cluster centers $\mathbf{w}_j$ will naturally satisfy the stochasticity constraints (12), (13), (14), and (15). This is the case when the Frobenius norm is adopted as the objective function.

4.2 Reducing the Computational Cost of the Factorization

This work presents the stochastic factorization as a general approach to reduce the number of states in an MDP. Of course, the main reason one would be interested in such a reduction is to save computational resources. One potential gain is in the amount of memory required to represent the model. Nevertheless, this paper is mainly concerned with the use of stochastic factorization as a strategy to reduce the *time complexity* of dynamic programming algorithms, that is, the number of arithmetic operations performed in the search for a decision policy. In this case, the computational cost of the factorization process itself is a concern. In particular, it only makes sense to resort to stochastic factorization if the number of operations involved in the factorization process is much smaller than the number of operations saved by replacing the original model with a compact one.

As discussed, the stochastic factorization problem is equivalent to nonnegative matrix factorization. Nonnegative matrix factorization has been a popular research topic in the last few years, and as a result the efficiency of the algorithms has been increasing steadily—in fact, nowadays it is possible to compute nonnegative factorizations of very large matrices in a matter of minutes (Esser et al., 2012; Bittorf et al., 2012). Recently, Thurau et al. (2011, 2012) have proposed efficient algorithms to approximately compute a convex hull that contains the rows of $\mathbf{M}$. They show that, by exploiting the extra structure in Problem 1, one can compute the approximation $\mathbf{DW} \approx \mathbf{M}$ much faster than algorithms that treat the problem as a conventional nonnegative factorization. There are also methods for the spectral unmixing problem that were specifically designed to be computationally efficient (Nascimento & Dias, 2004; Chang et al., 2006). Finally, if one interprets Problem 1 as a clustering task, it is possible to approximate very large matrices $\mathbf{M}$ by using some recently proposed methods (Boutsidis et al., 2010; Shindler et al., 2011).

In principle, any of the methods above can be used to compute the approximation $\mathbf{DW} \approx \mathbf{M}$ required by PISF. Some of them are able to solve the problem in time linear in the number of rows of $\mathbf{M}$ (Shindler et al., 2011; Thurau et al., 2012). Unfortunately, when it comes to Problem 1, this does not mean that the factorization will depend linearly on $|S|$. The problem is that the vectors $\mathbf{m}_i$ live in $\mathbb{R}^{|S|+1}$. This implies that the number of states in the MDP $\mathbf{M}$ defines not only the number of vectors $\mathbf{m}_i$ (the number of rows of $\mathbf{M}$), but also their dimension (the number of columns of $\mathbf{M}$). As a consequence, even the "linear" methods will run in $O(|S|^2)$ time.

Depending on the size of the MDP and on the computational resources available, a quadratic dependency on $|S|$ may be acceptable. There are many algorithms available in this case. For example, Shindler et al.'s (2011) clustering method can deliver an approximation $\mathbf{DW} \approx \mathbf{M}$ in $O(|S|^2|A|m)$ time, which corresponds to m iterations of the value iteration algorithm (Littman et al., 1995). There are also situations in which the transition matrices $\mathbf{P}^a$ are sparse, meaning that the execution of action a in state s_i can lead to a number of states $z \ll |S|$. In this case the factorization problem can be solved in time linear in $|S|$. The remaining of this section focuses on the worst case scenario, that is, the case in which $\mathbf{M}$ is *not* sparse and a quadratic dependency on $|S|$ is *not* acceptable.

4.2.1 BREAKING THE DOUBLE DEPENDENCY ON THE SIZE OF THE MDP

In order to make the stochastic factorization problem tractable for large MDPs, one needs to circumvent the fact that $|S|$ defines both the number of rows and the number of columns of $\mathbf{M}$. The strategy proposed in this section is to rewrite the problem in terms of a dissimilarity measure whose computation does not depend on the number of states in the MDP.

Let ϕ be a dissimilarity measure defined in $\mathbb{R}^{|S|+1} \times \mathbb{R}^{|S|+1}$ (such as a distance function, for example). This section will assume that the measure Φ previously introduced is induced by ϕ. For a few examples, let $\mathbf{A}$ and $\mathbf{B}$ be two arbitrary matrices of the same dimension. Then one can have, for instance,

$$\phi(\mathbf{a}_i, \mathbf{b}_i) \equiv \| \mathbf{a}_i - \mathbf{b}_i \|_{\mathrm{F}} \text{ and } \qquad \Phi(\mathbf{A}, \mathbf{B}) \equiv \textstyle\sum_i \phi(\mathbf{a}_i, \mathbf{b}_i) = \| \mathbf{A} - \mathbf{B} \|_{\mathrm{F}},$$

$$\phi(\mathbf{a}_i, \mathbf{b}_i) \equiv \| \mathbf{a}_i - \mathbf{b}_i \|_1 \text{ and } \qquad \Phi(\mathbf{A}, \mathbf{B}) \equiv \max_i \phi(\mathbf{a}_i, \mathbf{b}_i) = \| \mathbf{A} - \mathbf{B} \|_{\infty}, \quad (17)$$

$$\phi(\mathbf{a}_i, \mathbf{b}_i) \equiv \| \mathbf{a}_i - \mathbf{b}_i \|_2 \text{ and } \quad \Phi(\mathbf{A}, \mathbf{B}) \equiv \max_{\mathbf{x}, \|\mathbf{x}\|_2 = 1} \textstyle\sum_i \sqrt{[(\mathbf{a}_i - \mathbf{b}_i)\mathbf{x}]^2} = \max_{\mathbf{x}, \|\mathbf{x}\|_2 = 1} \| (\mathbf{A} - \mathbf{B})\mathbf{x} \|_2, \quad (18)$$

where $\| \cdot \|_2$ is the Euclidean norm. Based on the expressions above, it is clear that in order to minimize $\Phi(\mathbf{DW}, \mathbf{M})$ one can focus instead on minimizing $\phi(\sum_j d_{ij} \mathbf{w}_j, \mathbf{m}_i)$ for all i. One way of looking at $\sum_j d_{ij} \mathbf{w}_j \approx \mathbf{m}_i$ is to think of the rows $\mathbf{w}_j$ as a set of "prototypical" vectors that are representative of the dynamics of the MDP $\mathbf{M}$. Recalling that each row of $\mathbf{D}$ forms a convex combination, the element d_{ij} can be seen as the weight of representative vector $\mathbf{w}_j$ in the approximation of $\mathbf{m}_i$ (Cutler & Breiman, 1994; Keshava & Mustard, 2002). The core assumption of this section is that, in general, *d_{ij} should decrease with $\phi(\mathbf{m}_i, \mathbf{w}_j)$*. Although this is not necessarily true in an exact factorization, many approximation schemes rely implicitly or explicitly on such a premise (Hastie et al., 2002). One example is local kernel smoothing techniques such as the Nadaraya-Watson kernel-weighted estimator (Hastie et al., 2002, Chap. 6). In this case, the elements of $\mathbf{D}$ would be computed as:

$$d_{ij} = \frac{\exp(-\phi(\mathbf{m}_i, \mathbf{w}_j)/\tau)}{\sum_k \exp(-\phi(\mathbf{m}_i, \mathbf{w}_k)/\tau)}, \tag{19}$$

where τ controls the relative magnitude of the elements in one row of this matrix. More generally, d_{ij} should be computed based on a function ω that is non-increasing with respect to $\phi(\mathbf{m}_i, \mathbf{w}_j)$.

The assumption that d_{ij} decreases with $\phi(\mathbf{m}_i, \mathbf{w}_j)$ makes it possible to compute $\mathbf{D}$ based exclusively on ϕ, as in the example given in (19). It is also possible to compute $\mathbf{W}$ using only this dissimilarity measure. For example, Thurau et al. (2012) argue that, if one restricts the rows of $\mathbf{W}$ to be a subset of the rows of $\mathbf{M}$, the minimization of Φ as defined in (18) can be accomplished through the maximization of the volume of the simplex defined by the rows of $\mathbf{W}$. Based on concepts from distance geometry, the authors show that it is possible to efficiently compute the volume of the simplex defined by a candidate $\mathbf{W}$ using ϕ only. There are also several clustering algorithms that only require a distance matrix to work, never accessing the vectors $\mathbf{m}_i$ directly (Kaufman & Rousseeuw, 1990). Finally, one can derive simple heuristics that use ϕ to compute a matrix $\mathbf{W}$ that "covers" as well as possible the convex set defined by $\mathbf{M}$, ensuring that every row $\mathbf{m}_i$ is close to at least one representative vector $\mathbf{w}_j$. For example, one can adopt a constructive method that goes through all vectors $\mathbf{m}_i$ and successively adds new rows to the matrix $\mathbf{W}$ in order to guarantee that $\min_j \phi(\mathbf{m}_i, \mathbf{w}_j) < \sigma$ for all i, where σ is a predefined threshold. Notice that in this case the number of representative rows $\mathbf{w}_j$ will be automatically determined by the value of σ. This idea will be further explored in Section 4.2.2.

The obvious advantage of computing $\mathbf{DW} \approx \mathbf{M}$ based solely on ϕ is that the problem of reducing the computational cost of the factorization comes down to finding efficient ways of computing ϕ. Specifically, one can replace ϕ with an approximation $\tilde{\phi}$ whose computation requires fewer arithmetic operations. One way to define $\tilde{\phi}$ is to restrict the computation of $\phi(\mathbf{m}_i, \mathbf{m}_j)$ to a properly defined subset of the elements of $\mathbf{m}_i$ and $\mathbf{m}_j$—which corresponds to enforcing some degree of sparsity in $\mathbf{M}$ (see Mahoney, 2011). This section goes in another direction, though: it exploits the fact that each vector $\mathbf{m}_i$ is associated with a specific state-action pair which can often be represented by a feature vector whose dimension is much smaller than $|S|$.

From dynamic programming's perspective, a state s_k is nothing but an index $k \in \{1, 2, ..., |S|\}$. However, in an MDP describing a real decision problem, each state has a clear semantic interpretation. In general, a given state of the MDP can be represented by a set of features that are descriptive of the real state of the problem. Based on the state features, it is usually easy to define feature vectors that represent state-action pairs. Therefore, $S \times A$ can be thought of as a finite subset of a vector space; with a slight abuse of notation (s_i, a) will be used to refer to the vectors representing state-action pairs and $\dim(S \times A)$ will denote the number of components of (s_i, a).

In many cases, it is reasonable to assume that state-action features provide some information regarding the dynamics of the MDP. To be more precise, let $\tilde{\phi}$ be a dissimilarity measure defined in $S \times A$. The assumption is that, if state-action pair (s_k, a) is similar to state-action pair (s_l, b), then the associated transition probabilities and rewards should also be similar—that is, the distance between the corresponding vectors $\mathbf{m}_i$ and $\mathbf{m}_j$ should be small. Thus, one can use $\tilde{\phi}$ as a surrogate for ϕ. Note that this direct relationship between ϕ and $\tilde{\phi}$ will naturally happen when the MDP M is the result of the discretization of a model with continuous state space, as long as the functions defining the MDP's dynamics are reasonably smooth and the resolution of the discretization is sufficiently fine. Moreover, often only the *relative magnitude* of the distance matters for the proper functioning of the algorithms (Kaufman & Rousseeuw, 1990; Thurau et al., 2012). Therefore, an approximation $\tilde{\phi}((s_k, a), (s_l, b)) \approx C\phi(\mathbf{m}_i, \mathbf{m}_j)$, with $C > 0$, should suffice in many cases.

The strategy of replacing ϕ with a dissimilarity measure $\tilde{\phi}$ defined in the state-action space can result in significant computational savings when $\dim(S \times A) \ll |S| + 1$. This is akin to the well-known "kernel trick," in which an algorithm is rewritten in terms of inner products which are then replaced by an appropriately defined kernel (Schölkopf & Smola, 2002). The kernel trick allows one to work in very high-dimensional feature spaces without ever explicitly computing the features. Similarly, by adopting an algorithm based exclusively on ϕ, and then replacing the latter by $\tilde{\phi}$, one can compute the approximation $\mathbf{DW} \approx \mathbf{M}$ without ever manipulating the vectors $\mathbf{m}_i$ directly. This strategy can be used with any algorithm that computes $\mathbf{D}$ and $\mathbf{W}$ based on ϕ. As an illustration, the next section describes a concrete algorithm that does just that.

4.2.2 An Algorithm to Compute the Factorization of an MDP in Linear Time

The previous sections discussed several ways to address the stochastic factorization problem (Problem 1). Each approach has its advantages and drawbacks, and the decision about which method to adopt should take into account factors like the size of the problem, the level of accuracy required for the solution, and the computational resources available. This section describes in more detail a specific algorithm to compute the approximation $\mathbf{DW} \approx \mathbf{M}$. The objective is to provide an illustration of how some of the ideas described above can be implemented, and also make the discussion regarding computational and theoretical aspects of the factorization problem more concrete.

The proposed method, described in Algorithm 3, builds on the ideas discussed in Section 4.2.1. The strategy is to select a subset of the rows of $\mathbf{M}$ to form matrix $\mathbf{W}$, such that each row $\mathbf{m}_i$ has a representative vector $\mathbf{w}_j$ within a predefined neighborhood. Such a neighborhood is induced by a dissimilarity measure $\tilde{\phi}$ defined in $S \times A$. The mechanics of the method are very simple. It goes over each state-action pair of the MDP and computes their distance to the η-closest neighbors in the set E of state-action pairs already selected to be part of the model (line 7 of Algorithm 3). If the distance to the closest neighbor is above a predefined threshold σ, the corresponding row of $\mathbf{M}$ is added to $\mathbf{W}$ (lines 9 to 11). If not, the number of representative state-action pairs remains the same. Regardless of whether the model has grown or not, the η-closest neighbors are used to compute the elements in the z^{th} row of $\mathbf{D}$, where z is the index of the current state-action pair in M (lines 13 and 14). The elements d_{ij} can be computed by any function ω that does not increase with $\tilde{\phi}$ (see discussion in Section 4.2.1 and equation (19) for an example).

Algorithm 3 Stochastic-factorization computation

Require:
- MDP M — $\mathbf{M} \in \mathbb{R}^{|S||A| \times |S|+1}$ and the corresponding features (s_i, a)
- $\tilde{\phi} : (S \times A) \times (S \times A) \mapsto \mathbb{R}$ — similarity function
- $\omega : \mathbb{R} \mapsto \mathbb{R}$ — non-decreasing function
- $\sigma \in \mathbb{R}^+$ — neighborhood radius
- $\eta \in \mathbb{N}^*$ — number of neighbors in the approximation

Ensure: Factorization $\mathbf{DW} \approx \mathbf{M}$

1: $E \leftarrow \{(s_1, 0)\}$ ▷ E are the representative state-action pairs ($|E| = m$)
2: $\mathbf{D} \leftarrow \mathbf{0} \in \mathbb{R}^{|S||A| \times 1}$; $\mathbf{W} \leftarrow \mathbf{m}_1 \in \mathbb{R}^{1 \times |S|+1}$
3: **for** $i \leftarrow 1, 2, ..., |S|$ **do**
4:     **for** $a \leftarrow 1, 2, ..., |A|$ **do**
5:         $z \leftarrow (a-1) * |S| + i$ ▷ z is the row index of (s_i, a) in $\mathbf{M}$ (see (11))
6:         $h \leftarrow \min(\eta, |E|)$
7:         find the h nearest elements to (s_i, a) in E according to $\tilde{\phi}$; call the j^{th} closest pair $(s, b)_j$
8:         **if** $\tilde{\phi}((s,b)_1, (s_i, a)) > \sigma$ **then** ▷ A new representative state-action pair must be added
9:             $E \leftarrow E + \{(s_i, a)\}$
10:             $\mathbf{W} \leftarrow \begin{bmatrix} \mathbf{W} \\ \mathbf{m}_z \end{bmatrix}$ ▷ Add one row to $\mathbf{W}$
11:             $\mathbf{D} \leftarrow \begin{bmatrix} \mathbf{D} & \mathbf{0} \end{bmatrix}$ ▷ Add one column to $\mathbf{D}$
12:         **for** $j \leftarrow 1, 2, ..., h$ **do**
13:             $k \leftarrow$ index of $(s, b)_j$ in $\mathbf{W}$
14:             $d_{zk} \leftarrow \omega(\tilde{\phi}((s,b)_j, (s_i, a)))$
15:         **for** $j \leftarrow 1, 2, ..., |E|$ **do** $d_{zj} \leftarrow d_{zj} / \sum_{l=1}^{|E|} d_{zl}$ ▷ Make sure that $\mathbf{D}$ is stochastic

The parameter σ has a strong effect on the output of Algorithm 3: decreasing its value usually leads to a more accurate approximation $\mathbf{DW} \approx \mathbf{M}$, but it also increases the number m of representative state-action pairs (or, equivalently, representative vectors $\mathbf{w}_j$). The algorithm can go through the state-action pairs of the MDP in any order, and in the end every pair will have at least one representative counterpart within a distance of σ in the space $(S \times A, \tilde{\phi})$. However, since Algorithm 3 is a greedy method, the set of state-action pairs selected to be part of the model may change depending on the order in which the pairs are visited. The parameter η determines the number of representative vectors $\mathbf{w}_j$ used to approximate each $\mathbf{m}_i$, and can be seen as a device to control how local the approximation should be (this is akin to setting the parameter 'k' of a k-nearest neighbor approximation; see Hastie et al., 2002, Chap. 2, for an intuitive discussion). The parameter η also has a direct effect on the computational cost of the algorithm, as discussed next.

The most demanding operation in each iteration of Algorithm 3 is the computation of the η-nearest neighbors of the current state-action pair. There are several efficient algorithms available to perform this search, either exactly or approximately (Liu et al., 2005). The most popular exact method is to use a KD-tree, which takes $O(\dim(S \times A) m \log m)$ operations to be constructed and allows the search to be performed in $O(\eta \log m)$ time, on average (Friedman et al., 1977). Since Algorithm 3 performs $|S||A|$ iterations, its overall complexity is linear in $|S|$. Therefore, using this algorithm to build the approximation $\mathbf{DW} \approx \mathbf{M}$, and assuming that the number of iterations performed by PISF is much smaller than $|S|$, *the cost of the entire process of computing a decision policy depends only linearly on the number of states in the MDP*. This is the best one can do without assuming any extra structure in the model.

As for the space complexity of Algorithm 3, note that in an actual implementation the matrix $\mathbf{M}$ is not really necessary, and neither $\mathbf{W}$ nor $\mathbf{D}$ must be explicitly stored: $\mathbf{W}$ can be represented by an m-dimensional vector containing the indices of the representative state-action pairs and $\mathbf{D}$ can be a data structure with the $|S||A| \times \eta$ nonzero elements d_{ij}.

As mentioned, when Algorithm 3 terminates all state-action pairs of the MDP have at least one representative state-action pair within a neighborhood of radius σ defined in the space $(S \times A, \tilde{\phi})$. In order to extrapolate this guarantee to the performance of PISF, it is necessary to relate $\tilde{\phi}$ to ϕ. Let (s_k, a) and (s_l, b) be two state-action pairs associated with vectors $\mathbf{m}_i$ and $\mathbf{m}_j$, respectively. Then, if

$$\tilde{\phi}((s_k,a),(s_l,b)) < \sigma \Longrightarrow \phi(\mathbf{m}_i, \mathbf{m}_j) < \varepsilon, \tag{20}$$

it should be relatively straightforward to provide guarantees regarding PISF's solution. For example, if (17) is adopted and $\eta = 1$, one can resort to Proposition 3 to obtain such guarantees. There are specific scenarios where it should be easy to ensure that assumptions like (20) hold, such as for example when the MDP M results from the discretization of a continuous model. It may also be possible to derive guarantees similar to (20) based on knowledge about the problem, for example by looking at the transition equations describing the dynamics of the MDP. In general, though, it may be difficult to relate $\tilde{\phi}$ to ϕ. Note that it is trivial to modify Algorithm 3 to reflect the case in which $\tilde{\phi}$ is computed based on a subset of the elements of $\mathbf{m}_i$ and $\mathbf{m}_j$. In this case deriving guarantees analogous to (20) is considerably easier (see for example Mahoney, 2011).

Algorithm 3 relies on two premises: (i) given $\mathbf{M}$ and $\mathbf{W}$, making the elements d_{ij} inversely proportional to $\phi(\mathbf{m}_i, \mathbf{w}_j)$ results in a good approximation $\mathbf{DW} \approx \mathbf{M}$; ($ii$) it is possible to define a dissimilarity measure $\tilde{\phi} : (S \times A) \times (S \times A) \mapsto \mathbb{R}$ such that $\tilde{\phi}((s_k,a),(s_l,b)) \approx C\phi(\mathbf{m}_i, \mathbf{m}_j)$, with $C > 0$, where $\mathbf{m}_i$ and $\mathbf{m}_j$ are the rows of $\mathbf{M}$ associated with (s_k, a) and (s_l, b), respectively. It is important to point out that building an algorithm based on such assumptions is only one possible artifice to circumvent the computational cost of factoring an MDP. Other strategies may be possible, such as resorting to domain knowledge or developing methods that exploit some structural regularity of the MDP (see Section 5.3.3). In fact, there are scenarios where an exact factorization is readily available without the need for any computation (see Barreto, 2014, for an example). In any case, the next section shows empirically how Algorithm 3 can generate very good decision policies in some problems.

5. Computational Experiment

This section illustrates how PISF can be useful in practice with a real-world application of significant economical interest.

5.1 The Multicomponent-Replacement Decision Task

One of the big challenges faced by industry is the maintenance of assets over a long period of time. For example, a commercial airline or a cargo company must have an operational fleet, while a power company needs to maintain its electric power grid functioning at all times (Powell, 2007). In many cases, the maintenance of expensive equipments involves sums of money counted in the millions of dollars. In such situations, the decisions made during the maintenance activities may have an enormous economical impact.

Usually, an equipment such as a jet engine or an electric generator is composed of several components that degrade over time. If a critical component breaks, it has to be replaced immediately. It is often the case that such a maintenance operation has an associated *setup cost* which is independent of the number of components being replaced. This may be financial losses caused by the down time or costs associated with activities such as the disassembling of the equipment, delivery of new components, and displacement of specialists. Thus, in some circumstances, it may be advantageous to perform *opportunistic replacements* of functioning components to avoid future setup costs. As discussed, this trade-off between immediate and future costs is typical of decision making tasks.

The importance of the decision problem described above is attested by a huge body of literature originated in the 1960's and spanning the subsequent four decades (Barlow & Proschan, 1965; McCall, 1965; Pierskalla & Voelker, 1976; Sherif & Smith, 1981; Cho & Parlar, 1991; Dekker et al., 1997; Wang, 2002). The problem can be formalized as follows. Suppose that the asset of interest has n_c components and let $l_j \in \mathbb{N}^+$ denote the expected lifetime of component c_j measured in some discrete time unit. Then, each state s_i is a vector $\mathbf{s}_i \in \mathbb{N}^{n_c}$ whose j^{th} entry s_{ij} represents the remaining lifetime of component c_j. At each time step the remaining lifetime of c_j is decreased by one, and $s_{ij} = 0$ indicates that this component is no longer operational. Even if c_j has not reached the end of its useful life, it may fail with a given probability, which is a function of s_{ij} and possibly of the other components' remaining lifetimes s_{iu}. An inactive component c_j causes the entire asset to stop working, and if c_j is not replaced immediately a penalty of $-\infty$ takes place in the next transition. To avoid that, one must replace c_j, which incurs a cost of r_j dollars. On top of that, every replacement activity—joint or not—has an associated setup cost. The setup cost is composed of two terms, a fixed amount of R_s dollars plus an extra fee of R_f dollars which is only charged if a component has failed before reaching its expected lifetime. The extra fee covers the expenses of last-minute measures required by unexpected failures. Let action a be represented by a binary vector $\mathbf{a}_h \in \{0,1\}^{n_c}$ where $a_{hj} = 1$ indicates that component c_j should be replaced. The goal of the decision maker is to select an action $\mathbf{a}_h$ at each time step in order to minimize the expected discounted future cost.

As one can see, this problem suffers from a particularly severe version of the curse of dimensionality: not only does its state space grow fast with n_c, with $|S| = \prod_i (l_i + 1)$, but also the cardinality of the action space is an exponential function of this variable, since $|A| = 2^{n_c}$. Thus, even instances of the problem with only a small number of components already represent a difficult challenge for dynamic programming.

Given this scalability issue, researchers usually focus on particular cases of the multicomponent-replacement task whose structure can be exploited somehow. For example, it has been noted in the literature that when the failure cost R_f is not considered the state space of the multicomponent-replacement problem can be reduced in more than 50% by simply eliminating the states of S in which all components are functional (*i.e.*, $s_{ij} > 0$ for all j). Since one knows that the optimal action in the excluded states is to replace zero components, this modification does not change the optimal policy of the problem (see Sun et al., 2007; Arruda & Fragoso, 2011). Note though that when $R_f > 0$ it may be advantageous to carry out replacements even if no components have expired; as will be seen, in this case the reduction strategies cited above may fail.

Researchers and practitioners have also explored other constrained versions of the multicomponent-replacement task. For example, if the replacement costs r_j and the lifetimes l_j are the same for all components c_j, all permutations of a given vector $\mathbf{s}_i$ can be considered as being the same state, which reduces the state space considerably (Haurie & L'Ecuyer, 1982). Another simplification of the problem is to assume that the probability of component c_j failing is independent of the other components' remaining lifetimes s_{iu}. In this case, the components are related only through the "economical dependence" represented by the setup cost (Cho & Parlar, 1991). The resulting model, a factored MDP, can be solved orders of magnitude faster than a conventional MDP (see Section 5.3.3). It is also possible to reduce the action space of the multicomponent maintenance task by making some assumptions regarding the problem's dynamics (Xia et al., 2008).

Although the methods above can be very effective in particular instances of the maintenance task, they are not directly applicable to the most general version of the problem, in which components have different characteristics and depend on each other both economically and structurally. Hence, in practice industry often relies on simple "threshold policies" that replace all components with remaining lifetime above a given value (Haurie & L'Ecuyer, 1982; van der Duyn Schouten & Vanneste, 1990). Unfortunately, it is known that, in general, the optimal policy for the multicomponent-replacement problem does not lie in the space of threshold decision policies (Özekici, 1988; Xia et al., 2008).

5.2 Experimental Setup

The experiments described in this section were carried out assuming a general version of the maintenance task in which components interact with each other. The failure probability of component c_j was modeled as a linear function of the asset's general condition. Suppose the asset is in state $\mathbf{s}_i$ and the decision maker decides to perform action $\mathbf{a}_h$. Then, denoting the next state by $\mathbf{s}_k$, the probability of component c_j failing is:

$$P(s_{kj} = 0 \mid s_{ij} > 1, a_{hj} = 0) = f - \frac{(f - f_{min})(s_{ij} - 1)}{l_j - 1} + \hat{f}\,\frac{\sum_{u \neq j}(l_u - s_{iu})}{\sum_{u \neq j} l_u}, \tag{21}$$

where f, f_{min} and $\hat{f}$ are parameters of the model and it is assumed that $l_j > 1$ for all j. The motivation for equation (21) is that the probability of a given component failing should depend not only on the condition of the component itself, but also on the condition of all other components of the asset. This is in fact a generalization of a commonly used model of the problem; when $f = f_{min}$ and $\hat{f} = 0$, equation (21) reduces to the fixed failure probability often assumed in the literature (Sun et al., 2007; Arruda & Fragoso, 2011). In the experiments, equation (21) was considered with $f = 0.1$, $f_{min} = 0.01$, and $\hat{f} = 0.1$. Thus, if many components in the asset are about to expire, the probability of component c_j failing can more than double.

The formulation of the problem considered here is also general with respect to the characteristics of the individual components of the asset. Instead of assuming that all components have the same lifetime and cost, which seems unrealistic, the variables l_j and r_j were drawn from normal distributions with means $\mu_l = 10$ and $\mu_r = -10$ and common standard deviation $\sigma_{lr} = 3$ (the values sampled for l_j were rounded to the closest natural number and sampled again in case the result was smaller than 2). With this configuration, the *expected* value of the model's parameters coincide with the ones adopted by Sun et al. (2007). The constant term of the setup cost was fixed at $R_s = -10$ and the failure cost was set as $R_f = -5n_c$. As discussed above, when $R_f > 0$ it might happen that $\pi^*(\mathbf{s}_i) \neq \mathbf{0}$ even if $s_{ij} > 0$ for all j. This increases the effective size of the state space, since one cannot simply discard states without expired components.

The multicomponent-replacement task was modeled as a discounted problem with $\gamma = 0.999$ (as discussed in Puterman, 1994, in this case γ can be seen as a way of emulating the devaluation of money). For each value of $n_c \in \{2, 3, ..., 7\}$, 100 instances of the problem were randomly generated. The policy iteration algorithm was then used to find an optimal policy for the resulting MDPs (see Appendix A.3). As a means of comparison, policy iteration was also applied to the reduced version of the problem, in which a state $\mathbf{s}_i$ is removed from the MDP if and only if $s_{ij} > 0$ for all j. Note that such a modification requires the recalculation of the transition probabilities between the states that remain in the model (Arruda & Fragoso, 2011). Here, in order to accomplish this reduction, actions were replaced with their temporally extended counterparts, "options," giving rise to a "semi-MDP" (options are closed-loop policies with a well-defined termination condition and an initiation set; see Sutton et al., 1999, for details) .

In order to evaluate the heuristics usually adopted by industry, threshold policies using values ranging from 1 to 10 were considered. For each value of n_c, these policies were compared and the one providing the lowest expected cost was used in the comparisons with the other algorithms. Notice that a threshold policy using a value of 0 corresponds to the "naive" strategy of only replacing non-operational components. The performance of such a policy was used as a baseline in the comparisons.

As discussed in Section 4, PISF's configuration comes down to the definition of matrices $\mathbf{D} \in \mathbb{R}^{|S||A| \times m}$ and $\mathbf{W} \in \mathbb{R}^{m \times |S|+1}$. In the experiments of this section these matrices were computed by Algorithm 3. The following function was used as a similarity measure between state-action pairs:

$$\tilde{\phi}((\mathbf{s}_i, \mathbf{a}_h), (\mathbf{s}_k, \mathbf{a}_g)) = \begin{cases} \infty \text{ if } \mathbf{a}_h \neq \mathbf{a}_g, \\ \sum_{j=1}^{n_c} (1 - a_{hj}) |r_j| (s_{ij} - s_{kj})^2 \text{ otherwise}. \end{cases} \tag{22}$$

The intuition behind (22) is straightforward: two state-action pairs should be considered similar if, after the application of the actions to the corresponding states, the remaining lifetimes of the components are approximately the same (except for eventual failures). Note that the difference between the remaining lifetimes s_{ij} and s_{kj} is weighed by the magnitude of the cost r_j of replacing the j^{th} component. The other parameters of Algorithm 3 were defined as follows. The number of neighbors η used in the approximation was set to n_c, the function ω was defined as the constant $1/\eta$, and the neighborhood radius σ was varied in $\{200, 400, 600\}$. Since σ was the only parameter that varied across the experiments, the specific instances of PISF will be referred to as PISF-σ (see Appendix A.3 for more details regarding the implementation of PISF).

5.3 Results

Throughout this section, the following measure will be used to evaluate the algorithms:

$$\varphi(\mathbf{v}^{\pi}, \hat{\mathbf{v}} \mid \hat{S}) = \frac{1}{|\hat{S}|} \sum_{\{i \mid \mathbf{s}_i \in \hat{S}\}} \frac{v_i^{\pi} - \hat{v}_i}{|\hat{v}_i|}, \tag{23}$$

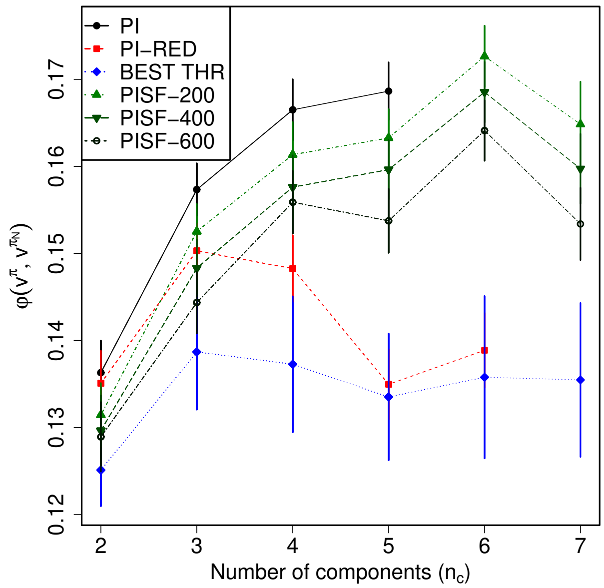


Figure 2: Expected gain on the multicomponent-replacement problem with respect to the naive decision policy. Error bars represent one standard error over 100 runs.

where $\mathbf{v}^{\pi}$ is the value function of the policy π returned by the algorithm under evaluation, $\hat{\mathbf{v}}$ is a reference value function, and $\hat{S} \subseteq S$ is the set of test states used in the evaluation (note that φ can take on negative values).

5.3.1 STANDARD SOLUTIONS

The two most natural ways of addressing the multicomponent-replacement problem is to use dynamic programming or to resort to the threshold policies normally adopted in practice. This section compares these methods with PISF. The performance measure used to evaluate the algorithms is the relative gain one should expect when using them instead of naively replacing a component only when it fails. Hence, the reference function $\hat{\mathbf{v}}$ in (23) is the value function of the naive policy, $\mathbf{v}^{\pi_N}$. When $n_c \leq 5$, it is possible to compute the value functions $\mathbf{v}^{\pi}$ and $\mathbf{v}^{\pi_N}$ exactly, and thus in this case the set $\hat{S}$ appearing in (23) is the entire state space S. However, for $n_c > 5$ computing $\mathbf{v}^{\pi}$ and $\mathbf{v}^{\pi_N}$ becomes infeasible in the computers used for the experiments. In this case $\hat{S}$ was composed of $10,000$ test states $\mathbf{s}_i$ sampled uniformly at random from S, and the corresponding values v_i^{π} and $v_i^{\pi_N}$ were approximated through Monte Carlo roll-outs of length $5,000$.[4]

Figure 2 compares the results of policy iteration (PI), policy iteration applied to the reduced version of the problem (PI-RED), the best threshold policy (BEST THR), and PISF using $\sigma = 200$, $\sigma = 400$, and $\sigma = 600$. One thing that immediately stands out in the figure is the fact that the performance of the optimal policies improves as the number of components n_c increases. This indicates that the financial losses of a company that does not make opportunistic replacements increase with the number of components in the asset.

4. The roll-outs were truncated at a point in which the value of the rewards (in the case of the current application, dollars) has already decreased in more than 99% (that is, $\gamma^{5000} < 0.01$).

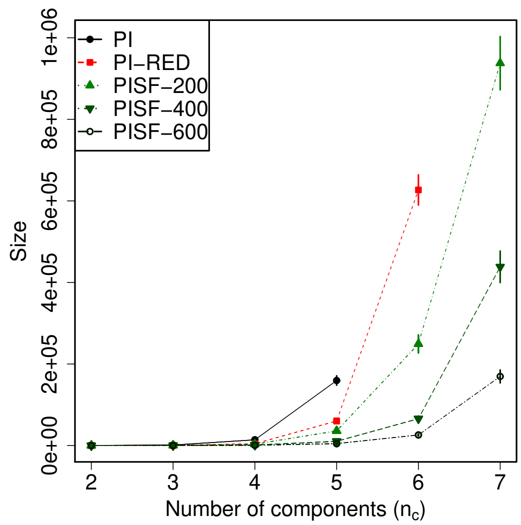
THR policies perform better than the naive policy in general. Notice though that the results shown in Figure 2 correspond to the performance of the *best* THR policy selected independently for each value of n_c. So, for example, while the average gain provided by the best THR policy when $n_c = 5$ is 13.36%, the average gain of the worst policy of this type is only 3.77%. This means that, in order to achieve the level of performance shown in Figure 2 in a real application, one cannot arbitrarily pick one specific THR policy. Rather, it is necessary to have a model of the problem and be willing to systematically compare all possible threshold values. Even in this case, the resulting policy will be suboptimal. For example, when $n_c = 5$, making optimal decisions increases the expected profit φ of the best THR policy in 27.8%, on average (see Figure 2). Considering the amounts of money often involved in maintenance activities, such a difference can represent significant monetary gains. This is a good illustration of the impact that dynamic programming may have in practice.

Unfortunately, dynamic programming does not scale well with the number of components in the multicomponent-replacement task. As mentioned above, in the experiments described here the optimal policy could not be computed for instances of the problem in which $n_c > 5$. Since in standard dynamic programming algorithms there is no easy way to control the amount of memory used, one is left with very few alternatives. In the case of the multicomponent-replacement task, one possibility is to eliminate states $\mathbf{s}_i > 0$ and apply dynamic programing to the reduced MDP. Such a strategy reduces the state space considerably, which makes it possible to solve MDPs one order of magnitude bigger (see Figure 2). Although the technique works well with the version of the problem considered by Arruda and Fragoso (2011), it may fail on the version of the problem considered here, in which the optimal policy may carry out opportunistic replacements in states without expired components. This is illustrated by the PI-RED curve in Figure 2, which clearly shows that the optimal policies of the reduced MDPs do not perform optimally in the original models.

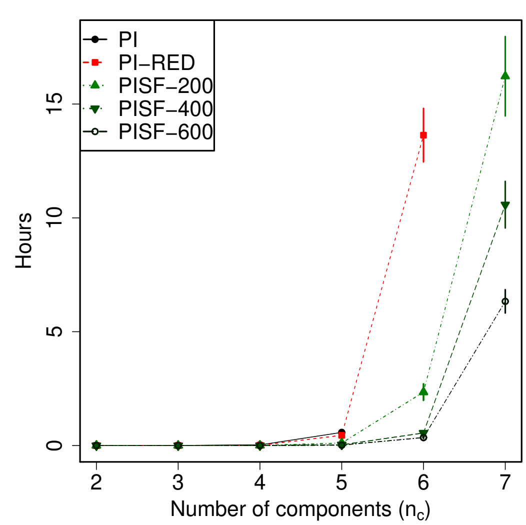
PISF represents an alternative between the two extremes of computing an optimal policy and resorting to simple heuristics such as the threshold policies. In contrast with conventional dynamic programming algorithms, PISF provides a practical mechanism to control the trade-off between the computational resources used and the quality of the resulting decision policy (the order m of the stochastic factorization, here indirectly defined by the neighborhood radius σ). This point is better illustrated when Figure 2 is analyzed in conjunction with Figure 3, which shows the size of the models generated by each algorithm and the associated time needed to solve them.

The dimension of the matrices processed by the algorithms during the value function computation is defined by the number of states in the corresponding Markov process. Thus, this number is a good measure of the algorithms' time and space complexities. In the case of policy iteration, what defines the size of the Markov processes is the number of states in the MDP; for PISF, such a number is m, the order of the stochastic factorization. The size of the Markov processes generated by each method is shown in Figure 3a. The figure makes it clear how the amount of memory required by the methods restricts their application. For example, if PI-RED was not run in the MDPs with $n_c = 7$ components, it is because the resulting Markov processes would be bigger than the largest model shown.

As expected, the size of the Markov processes generated by PISF decreases with σ. Take the MDPs with $n_c = 5$, for instance. In this case the average reduction on the original models provided by PISF was 60% when $\sigma = 200$, 88% when $\sigma = 400$, and 95% when $\sigma = 600$. When analyzing these numbers, one should keep in mind that the computational cost of evaluating a decision policy is cubic in the size of the Markov process. Thus, if the value functions are computed exactly, a



(a) Size of the Markov processes



(b) Run time (PISF's values include the time to compute the MDP factorization through Algorithm 3)

Figure 3: Size of the models generated by the algorithms and the associated time needed to compute a solution. Error bars represent one standard error over 100 runs.

reduction of 88% in the model's size translates to a decrease of $99,82\%$ on the number of arithmetic operations performed at each value-function computation.

Of course, in a real application the value function is seldom computed exactly. Besides, in order to use PISF one has to compute the factorization of the MDP. Even considering these factors, replacing PI with PISF can result in significant time savings. This is illustrated in Figure 3b, which shows the actual run times of the algorithms. For a concrete example, take again the MDPs with $n_c = 5$ components. In this case, adopting PISF-600 instead of PI results in a 97% reduction of computing time—which is equivalent to saying that the former algorithm is 37 times faster than the latter. To put this number in perspective, if it were possible to run PI in the MDPs with $n_c = 7$ components, finding a decision policy would take over 8 *days*. PISF-600 was able to compute an approximation in less than 5 *hours*.

But the reduction on the computational cost and memory usage provided by PISF are meaningless if the resulting decision policies perform poorly. Comparing Figures 2 and 3, one can clearly see that this is not the case in the multicomponent-replacement problem. For example, by replacing PI with PISF-400, the average reduction on computing time is over 90% for all values of n_c. In contrast, the expected decrease on the profit φ is below 6.2%. If one adopts PISF-600 instead of PISF-400, the reduction on the computational cost is at least 94%, for all values of n_c, while the associated losses are below 9.5%. This illustrates how PISF's performance degrades gracefully with the quality of the MDP's factorization, as indicated by the theoretical results presented in Section 3.3.

There is however a clear decrease on the expected gain provided by PISF when n_c increases from 6 to 7, as shown in Figure 2. One possible explanation for this is the fact that in the multicomponent-replacement task increasing the number of components in the asset increases not only the size of the state space S, but also the number of actions in A. This means that both the dimension *and* the number of matrices $\mathbf{P}^a$ increase exponentially with n_c, making it harder to "summarize" all the information in a single matrix $\mathbf{K}$. On top of that, and perhaps more important, by keeping the neighborhood radius σ fixed, the relative size of the models generated by PISF actually decreases. For example, for PISF-400, the average ratio $m/|S|$ is equal to 0.23 when $n_c = 3$, 0.12 when $n_c = 5$, and only 0.06 when $n_c = 7$. Thus, in order to keep the relative size of m fixed, σ must increase with n_c. In any case, even in the MDPs with $n_c = 7$ components PISF clearly outperforms the best THR policy, which is the only feasible alternative among the methods considered in this section. The next sections investigate other approximate solutions for the multicomponent-replacement problem.

5.3.2 Upper Confidence Bounds for Trees (UCT)

It is only possible to compute an optimal policy for the multicomponent-replacement problem if the number of components in the asset is below a certain threshold defined by the computational resources available. Above this threshold, the standard solution adopted by industry is to resort to simple heuristics. The previous section presented PISF as an alternative solution that allows for some control on the trade-off between the computational resources used and the quality of the resulting policy. It is only natural to ask whether other approximate methods would also be applicable in this case.

Two popular approaches for approximately solving an MDP are value-function approximation and state aggregation. These techniques are closely related to the stochastic-factorization trick, and will be discussed in Section 6. This section will focus on a set of methods collectively known as Monte-Carlo tree search (MCTS). MCTS methods are capable of computing approximate solutions for very large MDPs by combining tree search and random sampling (Browne et al., 2012). These methods have been receiving a lot of attention recently due to their enormous success in several applications, most notably in the game of Go (Bouzy & Helmstetter, 2003). MCTS algorithms use Monte Carlo roll-outs to estimate the return associated with the actions available at the current state. In order to do so, they build a tree, rooted at the current state, whose structure reflects the transitions performed during the roll-outs.

The main feature that distinguishes the different MCTS algorithms is the strategy used to select actions during the roll-outs. The most popular MCTS algorithm is Kocsis and Szepesvári's (2006) upper confidence bounds for trees (UCT). In UCT the action selection process is interpreted as a multi-armed bandit problem in which each arm corresponds to an action (Auer et al., 2002). Intuitively, UCT works by adding an "exploration bonus" to the values of actions that have been tried less often. One of the main parameters of the algorithm is the exploration constant C_p used to weigh the bonus against the value of actions (Kocsis & Szepesvári, 2006).

UCT has nice theoretical properties: given some $\varepsilon > 0$, it is guaranteed to return an ε-optimal action if the depth of the tree and the number of roll-outs are large enough (Kocsis & Szepesvári, 2006). Therefore, given enough computation, UCT will always be able to perform at the same level as PISF (and eventually better, if PISF's policy is not optimal). An interesting question in this case is how much computation is needed in practice for that to happen.

In order to answer this question, an experiment was carried out in which UCT and PISF were compared in the multicomponent-replacement problem. The algorithms were evaluated on the 100 MDPs with $n_c = 5$ components described in Section 5.2. As before, the performance of the resulting policies was contrasted with that of the naive policy π_N. The comparisons were based on a single roll-out starting from a state selected uniformly at random and independently for each MDP. More

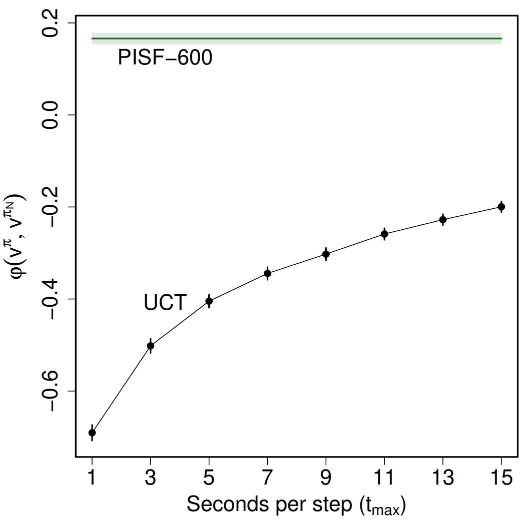


Figure 4: Expected performance on the multicomponent-replacement problem with respect to the naive decision policy. Error bars and shadow represent three standard errors over 100 runs.

specifically, the set of test states $\hat{S}$ appearing in (23) was composed of a single state s_i, and, as before, the corresponding values v_i^{π} and $v_i^{\pi_N}$ were approximated through a Monte Carlo roll-out of length $5,000$.[5]

As described above, before each action selection UCT builds a tree based on Monte-Carlo roll-outs. As the number of roll-outs grows, the quality of the value function approximation increases, but so does the computational cost of the algorithm. Thus, one way of evaluating UCT is to impose a time limit $t_{\max}$ on each iteration of the algorithm and check the performance of the resulting policy as a function of $t_{\max}$. Note that, given a fixed time budget $t_{\max}$, there is a trade-off between the number of roll-outs carried out and their length, and finding the right compromise between these two corresponds to finding the right balance between the bias and the variance of the value function approximation (Hastie et al., 2002). Here this issue was resolved by changing the maximum depth of the tree, $h_{\max}$, in the set $\{5, 50, 500, 5000\}$.[6] Also, UCT's exploration parameter C_p was varied in $\{10^0, 10^1, 10^2, 10^3\}$. Therefore, for each MDP and each value of $t_{\max}$, UCT was run 16 times—corresponding to all possible combinations of values for $h_{\max}$ and C_p—, and then the best result was selected and compared to that of the naive policy through (23). Figure 4 shows the average value of $\varphi(\mathbf{v}^{\pi}, \mathbf{v}^{\pi_N})$ as a function of $t_{\max}$. As a reference for comparison, the average result of PISF-600 on the same MDPs is also shown.

As shown in Figure 4, UCT performs *worse* than the naive policy π_N for $t_{\max} \leq 15$. Assuming the logarithmic trend shown in the figure will persist for $t_{\max} > 15$, UCT would need over 46 seconds per step to reach the level of performance of π_N. In order to find solutions comparable to those found by PISF-600, UCT would require around 115 seconds per step, or approximately 1.74 times the time needed by the former algorithm to compute a decision policy for the *entire* state space.

5. The policies were evaluated from a single test state only because the nature of the UCT algorithm makes it computationally impractical to carry out many roll-outs of length $5,000$. In any case, the small variance of (23) across the MDPs indicates that this decision did not have a strong effect on the comparisons.

6. Since $|(v^*_{\max} - v^*_{\min})/\bar{v}^*| < 0.002$ for all MDPs, where $\bar{v}^* = 1/|S| \sum_{i=1}^{|S|} v^*_i$, the value of leaf nodes was set to zero.

Note that, even though a time limit of $t_{\max} = 15$ seconds per step may not seem like much at first, the run time of UCT quickly escalates with the horizon of the problem. This becomes clear when one observes that a trajectory of $5,000$ steps with UCT using $t_{\max} = 15$ takes more than 20 hours, while computing the optimal decision policy for the same problem takes an average of 34 minutes.

It should be mentioned that the most basic version of UCT was used in the experiments of this section. Nowadays there are several extensions available in the literature (see for example Chaslot et al., 2008; Gelly et al., 2012; Keller & Eyerich, 2012). These strategies, when built on top of UCT, tend to improve its performance. That said, the fact that UCT needs orders of magnitude more computation than PISF to reach the same level of performance is not surprising, since the former only uses the MDP as a generative model—that is, the algorithm only samples from the conditional distributions represented by the transition matrices—while the latter fully exploits all the information available in the model.

5.3.3 FACTORED MDPS

The previous section illustrated how exploiting all the information available about a problem can be important for computational efficiency. This section shifts the focus of the discussion to another issue, namely that of the structure of the model. In particular, it describes a specific structured model known as *factored MDPs*.

In a factored MDP the transition dynamics can be described by a dynamic Bayesian network (DBN, Dean & Kanazawa, 1989; Boutilier et al., 1995). Let $\mathbf{s}^{(t)}$ and $\mathbf{s}^{(t+1)}$ denote, respectively, the state at the current time and at the next step. The transition graph of a DBN is a two-layer directed acyclic graph whose nodes are the variables in the set $\mathsf{S} = \{s_1^{(t)}, s_2^{(t)}, ..., s_n^{(t)}, s_1^{(t+1)}, s_2^{(t+1)}, ..., s_n^{(t+1)}\}$. An arc from $s_j^{(t)}$ to $s_i^{(t+1)}$ indicates that the value of the i^{th} variable at time $t+1$ depends on the value of the j^{th} variable at time t. Let $\rho(s_i^{(t+1)})$ denote the set of all nodes with an arc to $s_i^{(t+1)}$. The transition model of a factored MDP is given by $P(\mathbf{s}^{(t+1)}|\mathbf{s}^{(t)}) = \prod_i P(s_i^{(t+1)}|\rho(s_i^{(t+1)}))$. What allows a compact representation—and efficient solution—of a factored MDP is the assumption that the DBN is sparse, that is, $\rho(s_i^{(t+1)})$ is restricted to a small subset of S for all i. In addition, it is assumed that the reward is given by a summation of functions that depend on the status of only a few variables. When these assumptions hold, that is, when the MDP is truly factored, it is possible to represent the model's dynamics very compactly using structures like decision trees or algebraic decision diagrams (Boutilier et al., 1995; Hoey et al., 1999). Therefore, the number of states in the MDP remains the same; the computational gain comes from the fact that the state space is never explicitly enumerated.

At first, one might expect that the structure of a factored MDP would induce a value function with similar structure. If this were indeed the case, it would be possible to compute an optimal policy for a factored MDP much faster than dynamic programming algorithms that ignore the models' special structure. Unfortunately, Koller and Parr (1999) have shown that, in general, the value function of a factored MDP is not factored. Therefore, practical algorithms developed for factored MDPs only compute an approximate solution (see Guestrin et al., 2003, and references therein).

The factored MDP model captures the dynamics of many real-world decision-making tasks (Powell, 2007). In fact, it is safe to say that some MDPs solved using this technique are among the largest solved to date (Guestrin et al., 2003, see also Section 5.4). However, there are decision problems of great interest whose dynamics do not satisfy the assumptions of such models. The version of the multicomponent-maintenance problem addressed here is a good example: since the probability of a given component failing depends on the condition of all other components, the resulting DBN is not sparse (*cf.* equation (21)). But what happens if one ignores the fact that an MDP is not truly factored and applies the algorithms developed under such an assumption anyway?

To help answering this question, an experiment was performed in the following way. Let M be one of the MDPs with $n_c = 5$ components described in Section 5.2. For each component c_j and each value of $z \in \{1,2,3,4\}$, a set $\rho_z(c_j) \subset \{1,2,3,4,5\} - \{j\}$ containing z distinct elements was generated uniformly at random. Then, for each value of z, the transition matrices of M were regenerated with (21) replaced by

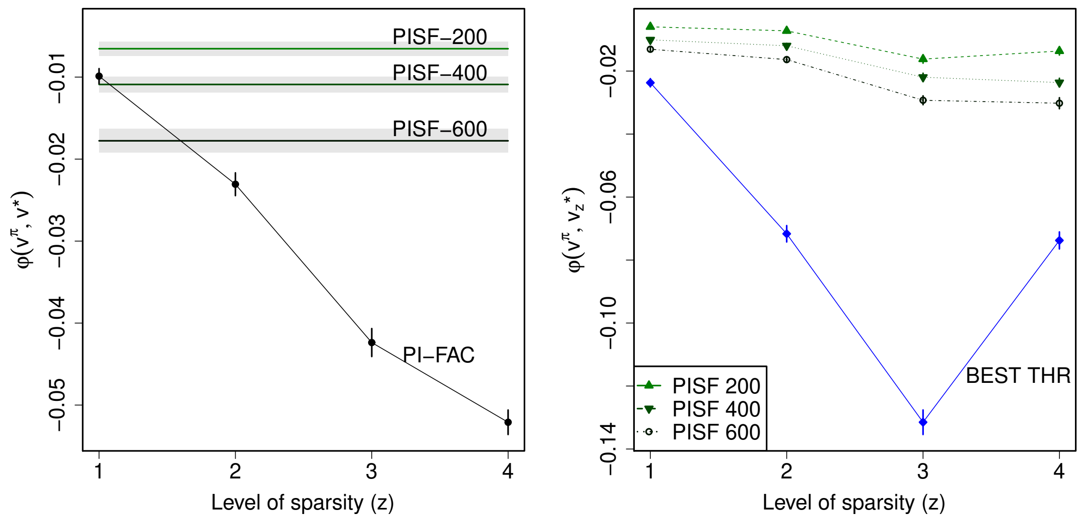
$$P(s_{kj} = 0 | s_{ij} > 1, a_{hj} = 0) = f - \frac{(f - f_{min})(s_{ij} - 1)}{l_j - 1} + \hat{f}\, \frac{\sum_{u \notin \rho_z(c_j)} (l_u - s_{iu}) + \sum_{u \in \rho_z(c_j)} (l_u - l_u/2)}{\sum_{u \neq j} l_u}, \tag{24}$$

giving rise to the MDP M_z. Using (24) corresponds to enforcing the sparsity of the DBN describing the components' failure probabilities. Note that, instead of completely ignoring the influence of the components c_u with $u \in \rho_z(c_j)$, their remaining lifetimes was replaced by $l_u/2$, the middle point of the range of possible values. This is a way of making the factored model closer to the original one without incurring in a denser DBN. After the transition matrices of M_z were generated, policy iteration was used to compute an optimal policy, π_z^* (note that this computation is exact). Finally, the performance of π_z^* *in the original MDP M* was compared to that of π^*, an optimal policy of M (thus, in this experiment the reference value function $\hat{\mathbf{v}}$ appearing in (23) is $\mathbf{v}^*$, and $\hat{S} = S$). This entire process was repeated for each one of the 100 MDPs with $n_c = 5$ components.

Figure 5a compares the performance of π_z^*, the policies returned by policy iteration on the artificially-factored MDPs (PI-FAC), with that of PISF-200, PISF-400, and PISF-600. Observe how the performance of PI-FAC degenerates as the level of sparsity z of the artificially-factored MDPs increases. This is expected, since the derived models deviate from the original ones as z approaches n_c. Since the computational cost of the algorithms developed for factored MDPs decreases with z, one has to trade performance for speed when choosing which interactions between variables to ignore. But the important point to note here is the fact that PI-FAC is outperformed by all instances of PISF when $z > 1$. Since π_z^* is an optimal policy of the factored MDPs, its performance is the best one can reasonably hope for when using algorithms that approximate it. In other words, a factored MDP algorithm using (24) will outperform PISF on the multicomponent-replacement task only if the approximation it computes "luckily" induces a policy that performs better than π_z^* in the original, non-factored, MDP.

This experiment shows that, by "pretending" that a non-factored MDP is factored, one may severely restrict the quality of the resulting policy, regardless of the specific algorithm chosen to solve the factored MDP. There is no reason to believe that this phenomenon is restricted to the multicomponent-replacement domain. Thus, there is a niche of problems that are too big to be solved by standard dynamic programming and cannot be reliably solved by algorithms developed for factored MDPs. For these problems, it might be a good alternative to resort to algorithms that exploit other types of structure, such as PISF.

This is not to say that the stochastic-factorization trick should be seen as an alternative to factored MDPs. In principle, the properties of an MDP being "factored" or "factorizable" are orthogonal to each other, and therefore both structures can potentially be exploited in conjunction. To illustrate this point, an experiment was carried out in which PISF and the THR policies were run in the factored MDPs M_z. Note the difference with respect to the previous experiment: while there the models M_z were considered as approximations of the MDPs M, here it is assumed that they *are* the true models. Thus, both PISF and the THR policies were compared to the optimal policies π_z^*. The



(a) Artificially-factored MDPs

(b) Truly-factored MDPs

Figure 5: Expected loss on the multicomponent-replacement problem with respect to the optimal decision policy. Error bars and shadows represent one standard error over 100 runs.

idea is to investigate how the structure of a transition matrix induced by a sparse DBN affects PISF's performance. Figure 5b shows the performance of PISF and the best THR policy on the factored MDPs M_z as a function of the level of sparsity z (*cf.* equation (24)). Although the performance of PISF degenerates slightly when z increases, the *relative* performance with respect to the best THR policy generally improves as the model gets more sparse.

This experiment illustrates how the stochastic-factorization trick can potentially be useful in the computation of a policy for a factored MDP, which may lead to the solution of very large sequential decision problems. Of course, in order to simultaneously exploit the "factored" and "factorizable" structures of a model, one has to apply the trick without explicitly manipulating the matrices involved. This constitutes an interesting extension of the current research, and is left as a suggestion for future work.

5.4 Discussion

This section described the application of PISF to a large problem of real interest. It was shown that, by using Algorithm 3 to compute $\mathbf{D}$ and $\mathbf{W}$, it is possible to handle MDPs whose exponentially large state and action spaces preclude the use of standard dynamic programming. The largest instance of the problem solved by PISF had $39,916,800$ states and 128 actions, or 359 times the size of the largest MDP solved by policy iteration. Note that in the maintenance problem studied here it is trivial to define heuristics that perform reasonably well, but in other applications this may not be true. For example, Dekker et al. (1996) point out that the multicomponent-replacement problem has a structure similar to that of other important decision-making tasks arising in production and inventory control. In such scenarios PISF can be an even more valuable tool.

It should be mentioned that the experiments of this section considered the classical scenario of dynamic programming, in which one seeks a decision policy defined over the entire state space S. Other formulations of the decision problem are possible. For example, in the field of automated planning it is often assumed that the decision maker must perform well from a small number of initial states only, and that at any state $s_i \in S$ only a small set of actions $A(s) \subset A$ is available (Ghallab et al., 2004; Geffner & Bonet, 2013). On top of that, it is sometimes assumed that the dynamics of the MDP are truly factored (Sanner, 2010). When all these assumptions hold, it is possible for the decision maker to perform well visiting only a small fraction of the state space, and, since the MDP's dynamics are factored, sometimes not even this small subset must be explicitly enumerated. Therefore, algorithms developed for this scenario can be applied to very large MDPs (Keller & Eyerich, 2012; Kolobov et al., 2012).

PISF is not a direct competitor of planning algorithms developed for the scenario above because it was not designed with such assumptions in mind. Since PISF is an offline method that computes a policy for the entire state space, it will never scale to MDPs as large as those handled by on-line methods that compute a policy on demand, such as UCT. Similarly, an MDP with truly factored dynamics should favor methods that exploit this structure. On the other hand, as the experiments of this section show, when the assumptions that underlie other planning algorithms do not hold, they can be largely outperformed by PISF.

In the end, what determines the success of a given algorithm is the suitability of its underlying assumptions to the scenario of interest. PISF has been developed for MDPs that are factorizable or nearly so—a structural regularity not exploited by any other dynamic programming algorithm. How often this structure arises in problems of interest and how well it can be exploited together with other assumptions, such as those usually considered in automated planning, are interesting questions to be addressed in future research.

6. Related Work

This section reviews some of the approaches that have been proposed to circumvent dynamic programming's curse of dimensionality, and discusses how these methods relate to the stochastic-factorization trick.

6.1 Value-Function Approximation

Perhaps the most straightforward approach to deal with a large MDP is to use a compact parametric representation of its value function. This is not a new idea; in fact, Bellman and Dreyfus explored the use of polynomials to approximate the value function in as early as 1959. Since then the theory has evolved a lot, and nowadays it is possible to find books that cover the subject in detail (Bertsekas & Tsitsiklis, 1996; Powell, 2007).

The main issue regarding the use of a parametric representation of the value function is the damaging effect it may have on dynamic programming algorithms. In particular, it is well known that the use of general approximators may cause instabilities or even the divergence of the algorithms (Boyan & Moore, 1995; Baird, 1995; Tsitsiklis & Roy, 1996, 1997). The most common strategy to deal with this problem is to restrict the structure of the approximator to compact representations with a linear dependence on the parameters (Tsitsiklis & Roy, 1997; Tadić, 2001; Schoknecht & Merke, 2003). Indeed, the use of linear approximators has led to a number of successful algorithms with good convergence properties (Tsitsiklis & Roy, 1996; Bradtke & Barto, 1996; Perkins & Precup, 2003; Lagoudakis & Parr, 2003; Sutton et al., 2008).

There is some evidence in the literature that the instability caused by some function approximators is related to their tendency to exaggerate the difference between two successive estimates of the value function (Thrun & Schwartz, 1993; Gordon, 1995; Ormoneit & Sen, 2002). For this reason, many researches have advocated the use of "conservative" function approximators that compute the value of a state as a weighted average of other states' values (Gordon, 1995; Tsitsiklis & Roy, 1996; Rust, 1997; Munos & Moore, 1999; Ormoneit & Sen, 2002; Szepesvári & Smart, 2004). Examples of such approximators include kernel averaging, linear interpolation, k-nearest neighbor, and some types of splines. In general, the combination of these approximators with dynamic programming leads to convergent algorithms (Gordon, 1995; Tsitsiklis & Roy, 1996).

Conservative function approximators are similar in nature to the stochastic-factorization trick. To see why this is so, consider the class of function approximators which Gordon (1995) called *averagers*. An averager approximates the value of a state by a convex combination of other states' values and possibly some predetermined constants. Given an MDP $M \equiv (S,A,P,R,\gamma)$, let $\bar{S}$ be a subset of the state space S, with $|\bar{S}| = m$, and let $\bar{\mathbf{v}} \in \mathbb{R}^m$ represent the values of the states in this subset. An averager would compute an approximation of the value function as

$$\tilde{v}_i = d_{i0}c_i + \sum_{j=1}^{m} d_{ij}\bar{v}_j, \tag{25}$$

with $c_i \in \mathbb{R}$, $d_{ij} \in \mathbb{R}^+$ and $\sum_{j=0}^{m} d_{ij} = 1$. Since in the approximation scheme above the values of all states can be determined from the values of the states in $\bar{S}$, only the latter must be updated during dynamic programming's iterative process.

For the sake of simplicity, suppose that there is an averager for which $d_{io} = 0$ for all $i \in \{1,2,...,|S|\}$. In this case, approximate dynamic programming using (25) corresponds to its exact version in a reduced MDP $\bar{M} \equiv (\bar{S},A,\bar{P},\bar{R},\gamma)$ where $\bar{P}^a(s_j|s_i) = \sum_{k=1}^{|S|} p_{ik}^a d_{kj}$ and $\bar{r}^a(s_i) = r^a(s_i)$ (Gordon, 1995, Thm. 4.1). This model represents a particular case of the stochastic-factorization trick in which there is a single matrix $\mathbf{D} \in \mathbb{R}^{|S|\times m}$ and one matrix $\mathbf{K}^a \in \mathbb{R}^{m\times|S|}$ for each $a \in A$ (Barreto et al., 2011). The dynamics of the reduced model are given by $\bar{\mathbf{P}}^a = \mathbf{K}^a\mathbf{D}$, where $\mathbf{K}^a$ is the matrix composed of the rows of the original transition matrix $\mathbf{P}^a \in \mathbb{R}^{|S|\times|S|}$ associated with the states $s_i \in \bar{S}$. By interpreting conservative approximators as a particular case of the stochastic-factorization trick, schemes similar to (25) can be thought of as approximating the MDP itself. Thus, both the definition of the approximator's architecture and the configuration of its parameters are converted into a well defined optimization problem, in which the objective is to find matrices $\mathbf{D}$ and $\mathbf{W}^a$ that minimize $\Phi(\mathbf{M}^a, \mathbf{D}\mathbf{W}^a)$ for all $a \in A$.

6.2 Model Reduction

Another way of handling large-scale decision-making problems is to find a compact representation of the associated MDP. In this case, the simplest idea is to aggregate states that share a common characteristic. There are many proposals of this type in the literature, and what distinguishes them is the criterion used to group states. For a detailed account of model approximation techniques, the reader is referred to the review provided by Li et al. (2006).

One of the earliest works on model approximation was that of Bertsekas and Castañon (1989), who propose aggregating and disaggregating states dynamically, during the value function computation, according to the residual left after a few applications of the Bellman operator. An alternative approach is to group states based on their associated transition probabilities and rewards. Following this line, Givan et al. (2003) suggest the notion of *stochastic bisimulation* as a criterion to aggregate states. Roughly speaking, two states are bisimilar if they have the same expected reward associated with each action, and the same transition probabilities to all groups of bisimilar states. This notion is closely related to Ravindran's (2004) concept of *homomorphism* between MDPs. Both bisimulation and homomorphism are principled criteria for state aggregation, and as such they guarantee the optimality of the decision policy "lifted" from the compact model. However, they are too restrictive to be applied in many real situations. Realizing that, several authors have proposed relaxed versions of these criteria (Dean et al., 1997; Ferns et al., 2006; Ravindran, 2004; Sorg & Singh, 2009).

Regardless of the specific criterion used to group states, state aggregation can be very naturally represented as a stochastic factorization. In this case, the rows of matrix $\mathbf{W}$ represent the dynamics associated with each group and matrix $\mathbf{D}$ has one nonzero element per row indicating the group to which each state-action pair belongs (note the flexibility of aggregating state-action pairs instead of states only). In this context, applying the stochastic-factorization trick corresponds to computing the probabilities of transitions between groups. A slightly more general approach for model reduction is to assume a *soft aggregation* of states, in which each state belongs to a group with given probability (Singh et al., 1995; Sorg & Singh, 2009). Soft aggregation is also naturally represented as a stochastic factorization by letting $\mathbf{D}$ have more than one nonzero element per row. In fact, it can be shown that Sorg and Singh's (2009) concept of *soft homomorphism* is equivalent to the particular case of the stochastic-factorization trick discussed in Section 6.1 (Barreto et al., 2011).

As one can see, the stochastic-factorization trick can serve as a useful formalism for thinking about state aggregation. For example, a hard aggregation of states corresponds to a matrix $\mathbf{D}$ in which each row contains a single '1' and all rows associated with a given state have the nonzero element in the same column. In a soft aggregation of state-action pairs both restrictions are removed. Also, a single application of the trick leads to a (soft) homomorphism, while successive applications lead to an aggregation/disaggregation scheme similar to Bertsekas and Castañon's (1989) approach. Perhaps more important, as with conservative approximators, the stochastic factorization turns the aggregation problem into a well defined optimization problem. This can provide a unifying framework for the analysis, comparison, and solution of the different versions of the aggregation problem.

7. Conclusion

The approach presented in this paper builds on a simple idea, called here the stochastic-factorization trick: given a stochastic factorization of a transition probability matrix, one can swap the factors of the multiplication to obtain another transition matrix, possibly much smaller than the original. This property can be exploited to reduce the number of states of a Markov process. Intuitively, the stochastic-factorization trick corresponds to creating a small number of representative states (or state-action pairs) and redirecting the transitions of the original model according to some similarity measure. Formally, this process can be posed as a well defined optimization problem.

The stochastic-factorization trick can be extended to an MDP in at least two ways: one can factor each Markov process that comes up in the search for a decision policy or factor the Markov processes associated with the actions of the problem before the search begins. If a single matrix $\mathbf{K}$ and a single vector $\bar{\mathbf{r}}$ are used in the latter, the PISF algorithm can be used to compute a decision policy for the problem at hand. PISF reduces the computational complexity of standard policy iteration from a cubic dependence on $|S|$ to a function that grows only linearly with the size of the MDP. PISF also enjoys nice theoretical guarantees, since it always converges to a decision policy whose performance improves with the quality of the MDP's factorization. As in general the factorization improves with its order, m, one can use this parameter to control the trade-off between the use of computational resources and the performance of the resulting policy.

In order to apply PISF to a decision-making problem, one must find an approximate factorization of the corresponding MDP, $\mathbf{M} \approx \mathbf{DW}$. One way to compute such a factorization is to see the rows of $\mathbf{W}$ as prototypical vectors that represent the dynamics of $\mathbf{M}$ and interpret the elements of $\mathbf{D}$ as a similarity measure between these vectors and the rows of $\mathbf{M}$. This way, $\mathbf{D}$ and $\mathbf{W}$ can be computed based on a dissimilarity measure defined in the problem's state-action space $S \times A$. Such a technique allows the approximation $\mathbf{M} \approx \mathbf{DW}$ to be computed in time linear in $|S|$. Therefore, the entire process of computing a decision policy with PISF will depend only linearly on the number of states of the MDP. Exploiting this fact, PISF was able to find approximate solutions for instances of an important decision task with more than 5 billion state-action pairs. The solutions found were considerably better than those found by a heuristic commonly adopted in practice.

Evidently, this paper does not exhaust the discussion on the stochastic-factorization trick and PISF. One subject that calls for further investigation is the development of alternative methods to efficiently compute matrices $\mathbf{D}$ and $\mathbf{W}$ (or the identification of scenarios where a factorization is available or easy to compute). Another promising research topic is the application of the stochastic-factorization trick to factored MDPs or other types of structured models. Finally, PISF may also be useful in the context of model-based reinforcement learning. In this case, instead of collecting sample transitions in order to estimate all parameters of an MDP $\mathbf{M}$, one can leverage the use of data by focusing on the prototypical state-action pairs represented in $\mathbf{W}$. After $\mathbf{W}$ has been determined, the elements of $\mathbf{D}$ can be computed based on some measure of similarity defined in $S \times A$, as done in the experiment presented in this paper. These topics constitute interesting directions for future research.

Appendix A. Modified Policy Iteration and Modified PISF

Throughout the paper, it is assumed that the value function $\mathbf{v}^{\pi}$ is computed exactly at each step of policy iteration. This facilitates the analysis of the theoretical properties and computational complexity of the algorithms. However, most of the ideas discussed extend naturally to the case in which $\mathbf{v}^{\pi}$ is only approximated.

A.1 Modified Policy Iteration

In Puterman and Shin's (1978) *modified policy iteration*, $\mathbf{v}^{\pi}$ is estimated through t applications of T^{π}. Thus, in this case each value function computation involves $O(|S|^2 t)$ operations. Decreasing t reduces the computational cost of evaluating a decision policy, but in general it also increases the number of policies that must be evaluated until convergence (Puterman, 1994). Note that when $t = 1$ one recovers the value iteration algorithm. Similarly, for $t \geq |S|$ one can simply compute $\mathbf{v}^{\pi}$ exactly by solving the associated linear system, which comes down to conventional policy iteration. Therefore, both value iteration and policy iteration can be seen as special cases of modified policy iteration.

The amount of memory used by modified policy iteration depends on the specific implementation adopted. In general, the memory usage is inversely proportional to the algorithm's effective run time. One extreme implementation strategy is to only store one vector corresponding to the current estimate of the value function, $\tilde{\mathbf{v}}^{\pi}$. Note though that in this case each multiplication $\mathbf{P}^{\pi}\tilde{\mathbf{v}}^{\pi}$ requires loading the rows of $\mathbf{P}^{\pi}$ from secondary memory (or computing them on demand), thus significantly increasing the algorithm's run time. One can speed up the algorithm by keeping the entire MDP loaded in memory at all times, therefore increasing the use of memory from $O(|S|)$ to $O(|S|^2|A|)$. An intermediate solution between these two extremes is to load (or compute) $\mathbf{P}^{\pi}$ before each value function computation, which leads to $O(|S|^2)$ memory usage.

A.2 Modified PISF

The definition of modified PISF is straightforward: the only change needed is to replace the exact computation of $\bar{\mathbf{v}}^{\pi}$ in line 6 of Algorithm 2 with t applications of $\bar{T}^{\pi}\bar{\mathbf{v}} = \bar{\mathbf{r}} + \gamma\bar{\mathbf{P}}^{\pi}\bar{\mathbf{v}}$. In this case, using PISF instead of modified policy iteration reduces the computational cost of each value function computation from $O(|S|^2 t)$ to $O(m^2 t)$. The reduction on memory usage depends on the strategy used to represent the models, as discussed in Appendix A.1.

It should be clear that using modified PISF with $t = 1$ corresponds to combining the stochastic-factorization trick with value iteration. Note though that, in terms of computational cost, this may not be the best alternative. As discussed in Appendix A.1, decreasing t tends to increase the number of iterations performed by PISF. Each iteration of PISF involves building matrix $\mathbf{D}^{\pi}$ and computing the multiplication $\mathbf{K}\mathbf{D}^{\pi}$ (lines 4 and 5 of Algorithm 2, respectively), which can be seen as "constructing" the operator $\bar{T}^{\pi}$ for the current π. Since the construction of $\bar{T}^{\pi}$ takes $O(|S|^2 m)$ operations and its application is only $O(m^2)$, one may be wasting computational effort by only applying this operator once.

A.3 Implementation Details

The experiments of Section 5 were performed using modified policy iteration and modified PISF (Appendices A.1 and A.2, respectively). Instead of fixing a value for t, the iterative value function computation was interrupted according to the stop criterion described in Puterman's (1994) Proposition 6.6.5, with $\varepsilon = 10^{-6}$. Policy iteration and PISF were run until two successive policies were identical, as shown in Algorithms 1 and 2. The matrices $\mathbf{P}^{\pi}$ of modified policy iteration were loaded before each value function computation, since this represents a compromise between memory usage and computational cost (see Appendix A.1). In the case of PISF only matrix $\mathbf{K}$ was kept in memory at all times; the matrices $\mathbf{D}^{\pi}$ were computed on demand at each iteration of the algorithm. All the statements in Section 5 regarding the algorithms' computational requirements refer to this specific implementation.

Acknowledgements

Part of this work was done while André Barreto was a postdoctoral fellow in the School of Computer Science at McGill University. The authors would like to thank Amir-massoud Farahmand for valid discussions, and also the anonymous reviewers for their suggestions to improve the paper. The experiments were run using computational resources made available by Compute Canada and *Calcul Québec*. Funding for this research was provided by *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES), the National Institutes of Health (grant R21 DA019800), the NSERC Discovery Grant program, and *Projets de Recherche en Équipe* (FQRNT).

References

Arruda, E., & Fragoso, M. D. (2011). Time aggregated Markov decision processes via standard dynamic programming. *Operations Research Letters*, *39*(3), 2576–2580.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, *47*(2-3), 235–256.

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 30–37.

Barlow, R. E., & Proschan, F. (1965). *Mathematical Theory of Reliability*. Wiley.

Barreto, A. M. S. (2014). Tree-based on-line reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Barreto, A. M. S., & Fragoso, M. D. (2011). Computing the stationary distribution of a finite Markov chain through stochastic factorization. *SIAM Journal on Matrix Analysis and Applications*, *32*, 1513–1523.

Barreto, A. M. S., Precup, D., & Pineau, J. (2011). Reinforcement learning using kernel-based stochastic factorization. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 720–728.

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

Bellman, R. E. (1961). *Adaptive Control Processes*. Princeton University Press.

Bellman, R. E., & Dreyfus, S. (1959). Functional approximations and dynamic programming. *Mathematical Tables and Other Aids to Computation*, *13*(68), 247–251.

Berry, M. W., Browne, M., Langville, A. N., Pauca, V. P., & Plemmons, R. J. (2007). Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics and Data Analysis*, *52*(1), 155–173.

Bertsekas, D. P. (1987). *Dynamic programming: deterministic and stochastic models*. Prentice-Hall.

Bertsekas, D. P. (1999). *Nonlinear Programming* (2nd edition). Athena Scientific.

Bertsekas, D. P., & Castañon, D. A. (1989). Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, *34*(6), 589–598.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Bittorf, V., Recht, B., Re, C., & Tropp, J. A. (2012). Factoring nonnegative matrices with linear programs. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1214–1222.

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995). Exploiting structure in policy construction. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 1104–1113.

Boutsidis, C., Zouzias, A., & Drineas, P. (2010). Random projections for *k*-means clustering. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 298–306.

Bouzy, B., & Helmstetter, B. (2003). Monte-Carlo Go developments. In *Advances in Computer Games*, pp. 159–174.

Boyan, J. A., & Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 369–376.

Bradtke, S. J., & Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, *22*(1/2/3), 33–57.

Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*, 1–43.

Chang, C.-I., Wu, C.-C., Liu, W., & Ouyang, Y. C. (2006). A new growing method for simplex-based endmember extraction algorithm. *IEEE Transactions on Geoscience and Remote Sensing*, *44*(10), 2804–2819.

Chaslot, G. M. J.-B., Winands, M. H. M., van den Herik, H. J., Uiterwijk, J. W. H. M., & Bouzy, B. (2008). Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, *4*, 343–357.

Cho, D. I., & Parlar, M. (1991). A survey of maintenance models for multi-unit systems. *European Journal of Operational Research*, *51*(1), 1–23.

Cohen, J. E., & Rothblum, U. G. (1991). Nonnegative ranks, decompositions and factorizations of nonnegative matrices. *Linear Algebra and its Applications*, *190*, 149–168.

Cutler, A. (1993). A branch and bound algorithm for constrained least squares. *Communications in Statistics—Simulation and Computation*, *22*(2), 395–321.

Cutler, A., & Breiman, L. (1994). Archetypal analysis. *Technometrics*, *36*(4), 338–347.

Dean, T., Givan, R., & Leach, S. (1997). Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 124–131.

Dean, T., & Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, *5*(2), 142–150.

Dekker, R., Wildeman, R. E., & van Egmond, R. (1996). Joint replacement in an operational planning phase. *European Journal of Operational Research*, *91*(1), 74–88.

Dekker, R., Wildeman, R. E., & van der Duyn Schouten, F. A. (1997). A review of multi-component maintenance models with economic dependence. *Mathematical Methods of Operations Research*, *45*, 411–435.

Denardo, E. V. (1967). Contraction mappings in the theory underlying dynamic programming. *SIAM Review*, *9*(2), 165–177.

Ding, C. H. Q., Li, T., & Jordan, M. I. (2010). Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(1), 45–55.

Esser, E., Mller, M., Osher, S., Sapiro, G., & Xin, J. (2012). A convex model for nonnegative matrix factorization and dimensionality reduction on physical space. *IEEE Transactions on Image Processing*, *21*(7), 3239–3252.

Ferns, N., Castro, P. S., Precup, D., & Panangaden, P. (2006). Methods for computing state similarity in Markov decision processes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 174–181.

Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, *3*(3), 209–226.

Gan, G., Ma, C., & Wu, J. (2007). *Data Clustering: Theory, Algorithms, and Applications*. ASA-SIAM Series on Statistics and Applied Probability. SIAM.

Geffner, H., & Bonet, B. (2013). *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Gelly, S., Kocsis, L., Schoenauer, M., Sebag, M., Silver, D., Szepesvári, C., & Teytaud, O. (2012). The grand challenge of computer Go: Monte Carlo tree search and extensions. *Communications of the ACM*, *55*(3), 106–113.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc.

Givan, R., Dean, T., & Greig, M. (2003). Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, *147*(1-2), 163–223.

Golub, G. H., & Loan, C. F. V. (1996). *Matrix Computations* (3rd edition). The Johns Hopkins University Press.

Gordon, G. J. (1995). Stable function approximation in dynamic programming. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 261–268.

Grippo, L., & Sciandrone, M. (2000). On the convergence of the block nonlinear Gauss-Seidel method under convex constraints. *Operations Research Letters*, *26*, 127–136.

Guestrin, C., Koller, D., Parr, R., & Venkataraman, S. (2003). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, *19*, 399–468.

Hartigan, J. A. (1975). *Clustering Algorithms*. John Wiley and Sons.

Hastie, T., Tibshirani, R., & Friedman, J. (2002). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.

Haurie, A., & L'Ecuyer, P. (1982). A stochastic control approach to group preventive replacement in a multicomponent system. *IEEE Transactions on Automatic Control*, *27*, 387–393.

Ho, N.-D., & van Dooren, P. (2007). Non-negative matrix factorization with fixed row and column sums. *Linear Algebra and Its Applications*, *429*(5–6), 1020–1025.

Hoey, J., St-Aubin, R., Hu, A. J., & Boutilier, C. (1999). SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 279–288.

Howard, R. (1960). *Dynamic Programming and Markov Processes*. MIT Press.

Kaufman, L., & Rousseeuw, P. J. (1990). *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley and Sons.

Keller, T., & Eyerich, P. (2012). PROST: Probabilistic planning based on UCT. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Keshava, N. (2003). A Survey of Spectral Unmixing Algorithms. *Lincoln Laboratory Journal*, *14*(1), 55–78.

Keshava, N., & Mustard, J. (2002). Spectral unmixing. *Signal Processing Magazine*, *19*, 44–57.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Proceedings of the European Conference on Machine Learning (ECML)*, pp. 282–293.

Koller, D., & Parr, R. (1999). Computing factored value functions for policies in structured MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1332–1339.

Kolobov, A., Mausam, & Weld, D. S. (2012). LRTDP versus UCT for online probabilistic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Lagoudakis, M. G., & Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, *4*, 1107–1149.

Lee, D. D., & Seung, H. S. (1997). Unsupervised learning by convex and conic coding. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 515–521.

Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, *401*, 788–791.

Lee, D. D., & Seung, H. S. (2000). Algorithms for nonnegative matrix factorization. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 556–562.

Li, L., Walsh, T. J., & Littman, M. L. (2006). Towards a unified theory of state abstraction for MDPs. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics*, pp. 531–539.

Lin, C.-J. (2007a). On the convergence of multiplicative update algorithms for nonnegative matrix factorization. *IEEE Transactions on Neural Networks*, *18*, 1589 – 1596.

Lin, C.-J. (2007b). Projected gradient methods for nonnegative matrix factorization. *Neural Computation*, *19*(10), 2756–2779.

Littman, M. L., Dean, T. L., & Kaelbling, L. P. (1995). On the complexity of solving Markov decision problems. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 394–402.

Liu, T., Moore, A. W., Gray, A., & Yang, K. (2005). An investigation of practical approximate nearest neighbor algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 825–832.

Mahoney, M. W. (2011). Randomized algorithms for matrices and data. *Foundations and Trends in Machine Learning*, *3*(2), 123–224.

McCall, J. J. (1965). Maintenance policies for stochastically failing equipment: A survey. *Management Science*, *11*, 493–524.

Munos, R., & Moore, A. (1999). Barycentric interpolators for continuous space & time reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1024–1030.

Nascimento, J. M. P., & Dias, J. M. B. (2004). Vertex component analysis: A fast algorithm to unmix hyperspectral data. *IEEE Transactions on Geoscience and Remote Sensing*, *43*, 898–910.

Ormoneit, D., & Sen, S. (2002). Kernel-based reinforcement learning. *Machine Learning*, *49 (2–3)*, 161–178.

Özekici, S. (1988). Optimal periodic replacement of multicomponent reliability systems. *Operations Research*, *36*, 542–552.

Paatero, P., & Tapper, U. (1994). Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, *5*, 111–126.

Perkins, T. J., & Precup, D. (2003). A convergent form of approximate policy iteration. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1595–1602.

Pierskalla, W. P., & Voelker, J. A. (1976). A survey of maintenance models: The control and surveillance of deteriorating systems. *Naval Research Logistics Quarterly*, *23*(3), 353–388.

Powell, W. B. (2007). *Approximate Dynamic Programming—Solving the Curses of Dimensionality*. John Wiley & Sons, Inc.

Puterman, M. L. (1994). *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

Puterman, M. L., & Shin, M. (1978). Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, *24*(11), 1127–1137.

Ravindran, B. (2004). *An Algebraic Approach to Abstraction in Reinforcement Learning*. Ph.D. thesis, University of Massachusetts, Amherst, MA.

Ravindran, B., & Barto, A. G. (2004). Approximate homomorphisms: A framework for non-exact minimization in Markov decision processes. In *Proceedings of the International Conference on Knowledge Based Computer Systems*.

Rust, J. (1997). Using randomization to break the curse of dimensionality. *Econometrica*, *65*(3), 487–516.

Sanner, S. (2010). Relational dynamic influence diagram language (RDDL): Language description.

Schoknecht, R., & Merke, A. (2003). Convergent combinations of reinforcement learning with linear function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1579–1586.

Schölkopf, B., & Smola, A. (2002). *Learning with Kernels*. MIT Press.

Sherif, Y. S., & Smith, M. L. (1981). Optimal maintenance models for systems subject to failure—a review. *Naval Research Logistics Quarterly*, *28*(1), 47–74.

Shindler, M., Wong, A., & Meyerson, A. W. (2011). Fast and accurate k-means for large datasets. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2375–2383.

Singh, S. P., Jaakkola, T., & Jordan, M. I. (1995). Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 361–368.

Sorg, J., & Singh, S. (2009). Transfer via soft homomorphisms. In *Autonomous Agents & Multiagent Systems/Agent Theories, Architectures, and Languages*, pp. 741–748.

Sun, T., Zhao, Q., & Luh, P. (2007). Incremental value iteration for time-aggregated Markov-decision processes. *IEEE Transactions on Automatic Control*, *52*, 2177–2182.

Sutton, R. S., Szepesvári, C., & Maei, H. R. (2008). A convergent $O(n)$ algorithm for off-policy temporal-difference learning with linear function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1609–1616.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

Szepesvári, C., & Smart, W. D. (2004). Interpolation-based Q-learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 791–798.

Tadić, V. (2001). On the convergence of temporal-difference learning with linear function approximation. *Machine Learning*, *42*(3), 241–267.

Thrun, S., & Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In *Proceedings of the Fourth Connectionist Models Summer School*, pp. 255–263.

Thurau, C., Kersting, K., Wahabzada, M., & Bauckhage, C. (2011). Convex non-negative matrix factorization for massive datasets. *Knowledge and Information Systems*, *29*, 457–478.

Thurau, C., Kersting, K., Wahabzada, M., & Bauckhage, C. (2012). Descriptive matrix factorization for sustainability adopting the principle of opposites. *Data Mining and Knowledge Discovery*, *24*(2), 325–354.

Tsitsiklis, J. N., & Roy, B. V. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning*, *22*, 59–94.

Tsitsiklis, J. N., & Roy, B. V. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*, 674–690.

van der Duyn Schouten, F. A., & Vanneste, S. G. (1990). Analysis and computation of (n,n)-strategies for maintenance of a two-component system. *European Journal of Operational Research*, *48*(2), 260–274.

Vavasis, S. A. (2009). On the complexity of nonnegative matrix factorization. *SIAM Journal on Optimization*, *20*, 1364–1377.

Wang, H. (2002). A survey of maintenance policies of deteriorating systems. *European Journal of Operational Research*, *139*(3), 469 – 489.

White, D. J. (1985). Real applications of Markov decision processes. *Interfaces*, *15*, 73–83.

White, D. J. (1988). Further real applications of Markov decision processes. *Interfaces*, *18*, 55–61.

White, D. J. (1993). A survey of applications of Markov decision processes. *The Journal of the Operational Research Society*, *44*(11), 1073–1096.

Whitt, W. (1978). Approximations of dynamic programs, I. *Mathematics of Operations Research*, *3*(3), 231–243.

Xia, L., Zhao, Q., & Jia, Q.-S. (2008). A structure property of optimal policies for maintenance problems with safety-critical components. *IEEE Transactions Automation Science and Engineering*, *5*(3), 519–531.