Modified Monotone Policy Iteration for Interpretable Policies in Markov Decision Processes and the Impact of State Ordering Rules

Sun Ju Lee[1], Xingyu Gong[1] and Gian-Gabriel Garcia[1*]

[1]H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, 30332, Georgia, United States of America.

*Corresponding author(s). E-mail(s): giangarcia@gatech.edu;
Contributing authors: julee@gatech.edu; xgong75@gatech.edu;

Abstract

Optimizing interpretable policies for Markov Decision Processes (MDPs) can be computationally intractable for large-scale MDPs, e.g., for monotone policies, the optimal interpretable policy depends on the initial state distribution, precluding standard dynamic programming techniques. Previous work has proposed Monotone Policy Iteration (MPI) to produce a feasible solution for warm starting a Mixed Integer Linear Program (MILP) that finds an optimal monotone policy. However, this prior work did not investigate the convergence and optimality of this algorithm, nor did they investigate the impact of state ordering rules, i.e., the order in which policy improvement steps are performed in MPI. In this study, we analytically characterize the convergence and optimality of the MPI algorithm, introduce a modified MPI (MMPI) algorithm, and show that our algorithm improves upon the MPI algorithm. To test MMPI numerically, we conduct experiments in two settings: 1) perturbations of a machine maintenance problem wherein the optimal policy is guaranteed to be monotone or near-monotone and 2) randomly generated MDPs. We propose and investigate 19 state ordering rules for MMPI based on each state's value function, initial probability, and stationary distribution. Computational results reveal a trade-off between computational time and optimality gap; in the structured machine maintenance setting, the fastest state ordering rules still

yield high quality policies while the trade-off is more pronounced in the random MDP setting. Across both settings, the random state ordering rule performs the best in terms of optimality gap (less than approximately 5% on average) at the expense of computational time.

1 Introduction

The Markov Decision Process (MDP) is a versatile modeling framework for formulating and solving sequential decision problems under uncertainty. MDPs have been formulated for a broad range of decision-making problems across many industries, including healthcare, manufacturing, supply chain, and energy (Boucherie and van Dijk, 2017). Subsequently, these models play an important role in improving the quality of decisions made across these industries. Furthermore, for high-stakes applications such as healthcare and criminal justice, the interpretability and intuitiveness of the resulting MDP-generated decision policy can play an important role in how the policy is implemented or whether it is implemented at all (Rudin, 2019; Rudin et al, 2022; McNealey et al, 2023).

While previous studies on the design of interpretable policies for MDPs vary in their definition of interpretable policies, many of these studies have shown that determining optimal interpretable policies can be computationally intractable for large-scale MDPs. For example, Petrik and Luss (2016) found that their problem reduces to a POMDP, and is NP-hard. Grand-Clément et al (2021) find that their problem reduces to the problem of finding an optimal classification tree, which implies that it is NP-hard. Garcia et al (2022) find that the total number of monotone policies grows as a binomial when either the state or action is fixed, but can grow at a larger rate if the two are dependent. Moreover, in both Garcia et al (2022) and Grand-Clément et al (2021), the authors find that the optimal interpretable policies are dependent on the initial state distribution, precluding the use of standard value iteration and policy iteration algorithms.

This research focuses on extending the work by Garcia et al (2022) on developing algorithms for optimizing over monotone policies. In this prior work, the authors formulate the problem as a Mixed Integer Linear Program (MILP) and propose a Monotone Policy Iteration (MPI) algorithm to produce a feasible solution to warm start the MILP, thereby reducing the computational effort needed to solve the problem. Because the optimal monotone policy is dependent on the initial state distribution, they performed policy improvement steps within MPI in decreasing order of initial state probabilities. However, they did not consider other state orderings for policy improvement, nor did they consider how this order can affect computational time or the optimality of the resulting policy. As such, we conduct an analysis of the convergence and optimality properties of the MPI algorithm, propose a Modified Monotone Policy Iteration (MMPI) algorithm to overcome shortcomings of MPI,

and design various state ordering rules in the MMPI algorithm. We show analytically that the MMPI algorithm possesses improved properties compared to the MPI algorithm and quantify through computational experiments how each ordering strategy affects the efficiency of the algorithm.

1.1 Literature Review

This research is most related to analysis of interpretable policies in MDPs and analysis of the policy iteration algorithm for MDPs.

Interpretable Policies for MDPs. Two of the most common types of interpretable policies for MDPs include ordering-based policies and tree-structured policies (McNealey et al, 2023). In ordering-based policies, the policy must obey an ordering or partial ordering over the states and/or actions. Common examples of ordering-based policies include threshold policies (Hu and Defourny, 2022; Alagoz et al, 2004; Shechter et al, 2008; Chen et al, 2018) and monotone policies (Kim et al, 2012; Kotas and Ghate, 2016; Garcia et al, 2022), with the latter being the focus of this work. Example applications include inventory management (Perera and Sethi, 2023), healthcare treatment planning (McNealey et al, 2023), and maintenance optimization problems (see §3.1). Early work on monotone policies for MDPs (Albright, 1979; Serfozo, 1976) and partially observable MDPs (Albright and Winston, 1979; Lovejoy, 1987) focused on identifying sufficient conditions which guarantee the existence of an optimal policy that is monotone. When an optimal monotone policy exists, standard methods (e.g., backward induction, policy iteration) can be used to solve for this policy. In general, however, the optimal policy may not be monotone; in such cases, optimizing the performance of an MDP over the monotone policies can be challenging. In recent work, Garcia et al (2022) formulate an exact solution method using MILP for solving this problem. To reduce the computational effort required for solving this problem, the authors present the MPI algorithm which modifies the classic policy iteration algorithm to generate a monotone policy, which can be used as an initial feasible solution (i.e., a warm start) to the MILP. We extend this prior research by analyzing how to prioritize states in each iteration of the algorithm via state ordering rules. Our analytical study shows that the ordering rule can play a role in both the convergence and optimality of the algorithm. Further, our numerical study demonstrates which types of ordering rules have superior performance.

Tree-structured policies for MDPs seek to exploit the inherently interpretable structure of decision trees. While several papers explore tree-structured policies for different types of sequential decision problems (Bravo and Shaposhnik, 2020; Ciocan and Mišić, 2022; Amram et al, 2022), the work by Grand-Clément et al (2021) focuses strictly on MDPs. In their work, Grand-Clément et al (2021) show that the problem of optimizing over tree-structured policies for a single decision epoch is NP-hard. As such, they design a value iteration algorithm for computing optimal MDP policies to only visit decision tree policies, ensuring that the resulting policy is a decision tree. While the

authors also recognize that the order over which states are visited in their algorithm plays an important role, they do not provide much guidance on how to select this ordering rule. As such, the findings of this research could potentially be applied to their setting as well.

Policy Iteration. Policy iteration is a classic dynamic programming algorithm for solving finite state infinite-horizon MDPs (Puterman, 2014, Ch. 6.4). The algorithm alternates between a policy evaluation and policy improvement step. The finite convergence and optimality of this algorithm were analyzed in Puterman and Brumelle (1979). Prior works have also made efforts to enhance the policy iteration algorithm. For example, Hastings and Mello (1973) and Grinold (1973) extend the work of MacQueen (1967) in developing action elimination procedures which speed up policy iteration. When sufficient conditions for the existence of an optimal monotone policy are met, White (1981) studies a variant of policy iteration in which policies are restricted to be monotone for the two-action case and Puterman (2014, Ch. 6.11.2) describes a similar algorithm for general action sets.

In recent years, modifications to policy iteration have been proposed for more complex variants of MDPs, such as certain types of reinforcement learning problems (Hu et al, 2018; Yu and Bertsekas, 2013; Topin et al, 2021; Bertsimas et al, 2022), approximate dynamic programming problems (Powell, 2016; McKenna et al, 2020), and robust MDPs (Satia and Lave, 1973; Li and Si, 2010; Kaufman and Schaefer, 2012; Sinha and Ghate, 2016). Notably, in all of these past works, there is an implicit or explicit assumption that the MDP under study satisfies the rectangularity property. That is, the action chosen in one state does not restrict the action that can be taken in another state. As such, the policy improvement step can be performed state-by-state, regardless of the order. In contrast, monotone policies (Garcia et al, 2022) and certain tree-structured policies (Grand-Clément et al, 2021) do not satisfy this assumption. This paper thus extends the policy iteration literature by providing analytical and numerical results for a class of MDP policies wherein the rectangularity property is not satisfied.

1.2 Contributions

Overall, the contributions of this work are as follows:

1. We examine the MPI algorithm proposed by Garcia et al (2022) and provide analytical results relating to the convergence and optimality of the algorithm. In particular, we prove that the algorithm does not always converge, and when it does converge, it may not be to the optimal monotone policy.
2. We propose a modified MPI (MMPI) algorithm which is guaranteed to converge to a policy with expected total discounted reward higher or equal to that of the policy returned by the MPI algorithm under certain conditions on the state ordering rule.
3. We formulate a scalable machine maintenance problem that is guaranteed to have an optimal monotone policy. This problem can be used as a test

bed for future analysis of monotone policies in MDPs or, more broadly, interpretable policies for MDPs.

4. We investigate variations of the MMPI algorithm on both machine maintenance MDPs and randomly generated MDPs to quantify the impact of the order by which to prioritize states in (a) the computation time required to converge to a feasible monotone policy and (b) the quality of the feasible monotone policy (i.e., the optimality gap). We discover state ordering rules in the MMPI algorithm that outperform the decreasing initial state probability method proposed by Garcia et al (2022) as well as the other ordering rules analyzed. We also propose hypotheses for why these ordering rules outperform others.

The remainder of this manuscript is organized as follows. In §2, we provide a technical background on MDPs, describe the problem of optimizing over monotone policies, and analyze the MPI and MMPI algorithms. In §3, we formulate a machine maintenance problem with stochastic repair options and show that it is guaranteed to have an optimal policy that is monotone. Next, we perform a computational study in §4 to explore the best-performing MMPI algorithm, using the machine maintenance problem of §3 as a testbed. We investigate the robustness of these results on randomly generated MDPs in §5. Finally, we provide concluding remarks in §6.

2 Methodology

In this section, we begin in §2.1 by providing a a brief technical introduction to discrete-time infinite horizon MDPs with finite state and action sets. In §2.2, we define monotone policies and provide known methods for obtaining an optimal or near-optimal monotone policy, including the MMPI algorithm. Finally, in §2.3, we analyze the convergence and optimality of the MMPI algorithm.

2.1 Markov Decision Processes

Formally, discrete-time infinite horizon MDPs with finite states and actions are defined by a tuple $(\mathcal{T}, \mathcal{S}, \mathcal{A}, r, P, \gamma)$, where $\mathcal{T} = \{1, ..., \infty\}$ is a set of discrete decision epochs, $\mathcal{S} = \{1, ..., S\}$ is a finite set of states that the system may occupy during the decision process, α is a distribution over the initial states, $\mathcal{A} = \{1, ..., A\}$ is a finite set of actions and $\mathcal{A}_s \subset \mathcal{A}$ is the set of actions that can be taken when the system occupies state $s \in \mathcal{S}$, P is the stationary transition probability matrix specifying $P(s'|s, a) = \mathbb{P}(s_{t+1} = s'|s_t = s, a_t = a)$, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function with $r(s, a)$ denoting the immediate reward obtained when the action a is taken while the system is in state s, and $\gamma \in [0, 1)$ is a discount factor applied to future rewards.

The decision process occurs as follows. First, the system is initialized according to a probability distribution α, with components $\alpha(s) = \mathbb{P}(s_1 = s)$. Then, for every decision epoch $t \in \mathcal{T}$, the decision-maker observes the system state s_t and performs an action a_t, incurring reward $r(s_t, a_t)$. Next, the system

evolves stochastically to state s_{t+1} according to the transition probability matrix P. The decision-maker's goal is to optimize a deterministic Markov policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes her expected total discounted reward in this decision process. Letting Π denote the set of all such policies, the optimal policy π^* satisfies

$$J^{\pi^*}(\alpha) = \max_{\pi \in \Pi} J^{\pi}(\alpha), \tag{1}$$

where the expected total discounted reward for any policy π under initial state distribution α is given by

$$J^{\pi}(\alpha) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, \pi(s_t)) \Big| s_1 \sim \alpha\right]. \tag{2}$$

Notably, $J^{\pi^*}(\alpha) = \sum_{s \in \mathcal{S}} \alpha(s) v^*(s)$, where the optimal value functions $v^*(s)$ specify the solution to the Bellman equations

$$v^*(s) = \max_{a \in \mathcal{A}} \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v^*(s') \right\} \text{ for all } s \in \mathcal{S}. \tag{3}$$

Many solution methods exist for solving (3), including linear programming, value iteration, and policy iteration. The policy iteration algorithm is described in Chapter 6.4 of Puterman (2014). We study a modification of this algorithm in our analysis of optimal monotone policies, which we describe next.

2.2 Monotone Policies

In this section, we describe monotone policies and algorithms for obtaining an optimal monotone policy for MDPs. First, we define monotone policies.

Definition 1 (Monotone Policy) Suppose that the state space $\mathcal{S}$ and action space $\mathcal{A}$ are totally ordered sets. Then, a policy π is a monotone policy if $s \geq s'$ implies that $\pi(s) \geq \pi(s')$.

Practically speaking, a monotone policy specifies actions that increase in severity with the state. Letting $\Pi^M \subset \Pi$ denote the set of all monotone policies, an optimal monotone policy $\pi^M \in \Pi^M$ satisfies

$$J^{\pi^M}(\alpha) = \max_{\pi \in \Pi^M} J^{\pi}(\alpha). \tag{4}$$

An exact MILP solution to (4) is given by Garcia et al (2022). Letting $M_{s,a}$ denote large constants for each $s \in \mathcal{S}$ and $a \in \mathcal{A}$, this formulation is given by:

$$\max_{\boldsymbol{v}, \boldsymbol{x}} \quad \sum_{s \in \mathcal{S}} \alpha(s) v(s) \tag{5a}$$

$$\text{subject to:} \quad v(s) \leq r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a)v(s') + M_{s,a}\,(1 - x(s,a)) \quad \text{for all } s \in \mathcal{S}, a \in \mathcal{A} \tag{5b}$$

$$x(s,a) \leq \sum_{a' \geq a} x(s+1, a') \text{ for all } s \in \mathcal{S} \setminus S, a \in \mathcal{A} \tag{5c}$$

$$x(s,a) \in \{0,1\} \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}. \tag{5d}$$

Given any optimal solution $(\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{x}})$ of (5), an optimal monotone policy is given by setting $\pi^M(s) = a$ wherever $\tilde{x}(s,a) = 1$. The dual formulation of formulation (5) is given by

$$\max_{\boldsymbol{x},\boldsymbol{y}} \quad \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} r(s,a) y(s,a) \tag{6a}$$

$$\text{subject to:} \quad \sum_{a \in \mathcal{A}} y(s,a) - \gamma \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} P(s|s',a') y(s',a') = \alpha(s) \text{ for all } s \in \mathcal{S} \tag{6b}$$

$$\sum_{a \in \mathcal{A}} x(s,a) = 1 \text{ for all } s \in \mathcal{S} \tag{6c}$$

$$y(s,a) \leq M_{s,a} x(s,a) \text{ for all } s \in \mathcal{S},\ a \in \mathcal{A} \tag{6d}$$

$$x(s,a) \leq \sum_{a' \geq a} x(s+1, a') \text{ for all } s \in \mathcal{S} \setminus S, a \in \mathcal{A} \tag{6e}$$

$$y(s,a) \geq 0 \text{ for all } s \in \mathcal{S}, a \in \mathcal{A} \tag{6f}$$

$$x(s,a) \in \{0,1\} \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}. \tag{6g}$$

Note that the dual formulation (6) is derived by incorporating the binary variables $\boldsymbol{x}$ into the standard dual formulation for solving infinite horizon MDPs via linear programming (Puterman, 2014). That is, $x(s,a) = 0$ implies that the state-action occupancy measure $y(s,a) = 0$, and otherwise, $y(s,a) > 0$.

Critically, we remark that formulations (5) and (6) may be computationally intractable for large-scale MDPs. As such, Garcia et al (2022) design the MPI algorithm, which is detailed in Algorithm 1, to generate a feasible monotone policy that can be used to warm-start their MILP formulation. We now briefly describe the MPI algorithm.

The MPI algorithm is initialized with a state ordering rule $\{\sigma_n\}_{n \geq 1}$ where the mapping $\sigma_n : \mathcal{S} \to \mathcal{S}$ generates a strict ordering over the states. That is, σ_n describes the order in which states are visited in the policy improvement procedure in iteration n of the algorithm. For example, $\sigma_n(1) = 3$ implies that in iteration n, state 3 is visited first. Note that the state ordering rule σ_n does not have to be known for all $n \geq 1$ in advance, but can be computed at each iteration. In addition to $\{\sigma_n\}_{n \geq 1}$, MPI also requires an initial state value function $\boldsymbol{v}_0$ and a termination threshold $\epsilon > 0$. Then, in each iteration n, MPI loops through the states in the order given by σ_n. For each state $s = \sigma_n(i)$,

Algorithm 1 Monotone Policy Iteration

Require: $\{\sigma_n\}_{n\geq 1}$, $\boldsymbol{v}_0$, $\epsilon > 0$, $n = 0$
Ensure: $\sigma_n : \{1, ..., S\} \to \{1, ..., S\}$ is a one-to-one mapping for each n
while $n = 0$ or $\|\boldsymbol{v}_n - \boldsymbol{v}_{n-1}\| > \epsilon$ **do**
  $n \Leftarrow n + 1$
  $\bar{\mathcal{S}} \Leftarrow \emptyset$
  for $i = 1$ to S **do**
    $s \Leftarrow \sigma_n(i)$
    $\mathcal{S}^- \Leftarrow \{s' \in \bar{\mathcal{S}} : s' < s\}$
    $\mathcal{S}^+ \Leftarrow \{s' \in \bar{\mathcal{S}} : s' > s\}$
    if $\mathcal{S}^- = \emptyset$ **then**
      $A_{\min}(s) \Leftarrow 1$
    else
      $A_{\min}(s) \Leftarrow \pi_n(\max \mathcal{S}^-)$
    end if
    if $\mathcal{S}^+ = \emptyset$ **then**
      $A_{\max}(s) \Leftarrow A$
    else
      $A_{\max}(s) \Leftarrow \pi_n(\min \mathcal{S}^+)$
    end if
    $\mathcal{A}_n(s) \Leftarrow \{a \in \mathcal{A} : A_{\min}(s) \leq a \leq A_{\max}(s)\}$
    $\pi_n(s) \Leftarrow \arg\max_{a \in \mathcal{A}_n(s)} r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v_{n-1}(s')$
    $\bar{\mathcal{S}} \Leftarrow \bar{\mathcal{S}} \cup \{s\}$
  end for
  Perform policy evaluation on π_n to obtain $\boldsymbol{v}_n$
end while
return $\pi \Leftarrow \pi_n$

for $i = 1, \ldots, S$, the algorithm uses two sets, $\mathcal{S}^-$ and $\mathcal{S}^+$, to keep track of the states previously visited in iteration n. Specifically, previously visited states having a lower state index are stored in the set $\mathcal{S}^-$ and those with a higher state index are stored in the set $\mathcal{S}^+$. Using these sets, the algorithm constructs a set of feasible actions $\mathcal{A}_n(s)$, i.e., actions available to be assigned to $\pi_n(s)$ which ensure that the resulting policy π_n is monotone. In particular, the set $\mathcal{A}_n(s)$ is defined as $\{a \in \mathcal{A} : A_{\min}(s) \leq a \leq A_{\max}(s)\}$, where $A_{\min}(s)$ denotes the action with the smallest index allowed to be chosen for $\pi_n(s)$, and $A_{\max}(s)$ denotes the action with the largest index allowed to be chosen for $\pi_n(s)$. For example, if $s - 1 \in \mathcal{S}^-$, i.e., state $s - 1$ has been previously visited in the current iteration, then $A_{\min}(s) = \pi_n(s - 1)$ since $\pi_n(s) \geq \pi_n(s - 1)$ in order for π_n to be monotone. Then, $\pi_n(s)$ is assigned according to the action which maximizes the Bellman update operator applied to the value function from the previous iteration, $\boldsymbol{v}_{n-1}$, over the set of feasible actions $\mathcal{A}_n(s)$. MPI terminates when the distance between subsequent value functions exceeds the specified termination threshold ϵ, i.e., when $\|\boldsymbol{v}_n - \boldsymbol{v}_{n-1}\| > \epsilon$. In the next section, we provide an analytical characterization of this algorithm.

2.3 Analysis of Monotone Policy Iteration

While Garcia et al (2022) utilize the MPI algorithm, they do not analyze the properties of this algorithm. In this section, we provide several results relating to the convergence and optimality of MPI.

Proposition 1 *Suppose that there exists an integer n^* such that for all $n \geq n^*$,*

$$\pi_{n-1}(s) \in \mathcal{A}_n(s) \text{ for all } s \in S, \tag{7}$$

i.e., for each state, the policy found in the previous iteration of MPI is in the set of available actions in the current iteration. Then, MPI will converge in finitely many iterations. Otherwise, MPI may not converge.

Proof First, we show the case where the algorithm is guaranteed to converge. Consider any $n > n^*$ and let $\boldsymbol{\pi}_{n-1}$ and $\boldsymbol{\pi}_n$ be successive policies generated by Algorithm 1. Let $\boldsymbol{\pi}_n$ satisfy

$$\pi_n(s) \in \underset{a \in \mathcal{A}_n(s)}{\arg\max} \left\{ r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{n-1}(s') \right\}.$$

Then, by assumption, $\pi_{n-1}(s) \in \mathcal{A}_n(s)$ for all $s \in \mathcal{S}$ and

$$\boldsymbol{r}_{\pi_n} + \gamma P_{\pi_n} \boldsymbol{v}_{n-1} \geq \boldsymbol{r}_{\pi_{n-1}} + \gamma P_{\pi_{n-1}} \boldsymbol{v}_{n-1} = \boldsymbol{v}_{n-1}$$

where the right hand side of the inequality comes from the policy evaluation step. Rearranging terms yields

$$\boldsymbol{r}_{\pi_n} \geq (\boldsymbol{I} - \gamma P_{\pi_n}) \boldsymbol{v}_{n-1} \Rightarrow \boldsymbol{v}_n = (\boldsymbol{I} - \gamma P_{\pi_n})^{-1} \boldsymbol{r}_{\pi_n} \geq \boldsymbol{v}_{n-1},$$

where $(I - \gamma P_{\pi_n})^{-1}$ exists for $0 \leq \gamma < 1$ since γP_{π_n} is a bounded linear transformation with spectral norm < 1. Thus, the values $\boldsymbol{v}_n$ of successive policies generated by MPI are nondecreasing. Since there are finitely many deterministic stationary monotone policies, we will eventually have $\boldsymbol{v}_n = \boldsymbol{v}_{n-1}$ after a finite number of iterations which satisfies the termination criteria.

Now, suppose that instead, equation (7) does not hold. We construct an MDP to show that in this case, the algorithm may not converge. Consider the MDP in Figure 1and let the inputs to MPI be $\boldsymbol{v}_0 = (0, 10, 0)$ and state ordering rule

$$\sigma_n = \begin{cases} \{1,2,3\} \text{ if } n \text{ odd} \\ \{2,1,3\} \text{ if } n \text{ even.} \end{cases}$$

Then, the first iteration of MPI yields $\pi_1 = (2, 2, 2)$. Then, the algorithm gives

$$\pi_n = \begin{cases} (2,2,2) \text{ if } n \text{ odd} \\ (1,1,1) \text{ if } n \text{ even,} \end{cases}$$

with corresponding value functions

$$\boldsymbol{v}_n = \begin{cases} (0,10,0) \text{ if } n \text{ odd} \\ (10,0,0) \text{ if } n \text{ even.} \end{cases}$$

We can verify that equation (7) does not hold: for $n \geq 2$ and $s \in \{\sigma_n(2), \sigma_n(3)\}$, $\pi_{n-1}(s) \notin \mathcal{A}_n(s)$. Thus, the algorithm does not converge for any

$$\epsilon < \left\| [0 \ 10 \ 0]^\top - [10 \ 0 \ 0]^\top \right\|.$$

□

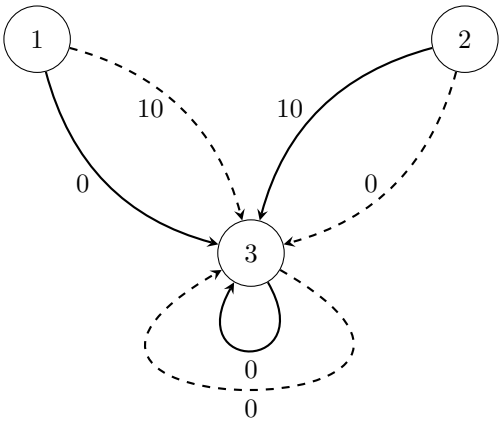

Figure 1 An MDP for which the MPI algorithm may not converge or may not converge to the optimal monotone policy. The solid (dashed) lines represent transitions corresponding to action 1 (action 2) which occur with probability 1, and are labeled with the corresponding rewards.

Proposition 1 implies important practical considerations for using the MPI algorithm: it may be necessary to specify stronger termination criteria, such as a maximum number of iterations or a tie-breaking rule on the policy to avoid cycling. We now show that even when MPI converges, it is not guaranteed to converge to an optimal monotone policy.

Proposition 2 *The MPI algorithm is not guaranteed to converge to an optimal monotone policy.*

Proof We show this proof by counterexample. Consider the MDP in Figure 1. If the initial distribution is $\alpha = (0.75, 0.25, 0)$, then there exists a unique optimal monotone policy $\pi^M = (2, 2, 2)$ with expected total discounted reward $J^{\pi^M}(\alpha) = 0.75(10) + 0.25(0) = 7.5$. Suppose that MPI is carried out with $\sigma_n = \{2, 1, 3\}, n \geq 1$. Starting with $\sigma(1) = 2$, we have $\pi_1(2) = 1$. Now, for $\sigma(2) = 1$, we have $\bar{\mathcal{S}} = \{2\}$, which gives $\mathcal{S}^- = \emptyset$ and $\mathcal{S}^+ = \{2\}$. Then $\mathcal{A}_{\min}(1) = 1$ and $\mathcal{A}_{\max}(1) = \pi_1(2) = 1$, which gives $\pi_1(1) = 1$. Following the same calculations for $n = 2$, we have $\pi_2 = \pi_1$, and MPI converges to the policy $\pi^1 = (1, 1, 1)$ with expected total discounted reward $J^{\pi^1}(\alpha) = 0.75(0) + 0.25(10) = 2.5$, which is sub-optimal as $J^{\pi^1} < J^{\pi^M}$. Hence, MPI may not always converge to an optimal monotone policy. □

In the following result, we identify conditions under which the policy resulting from the MPI algorithm is near-optimal. We first define the notion of a δ-optimal monotone policy.

Definition 2 (δ-optimal monotone policy) We say that a monotone policy π is a δ-optimal monotone policy if

$$J^{\pi}(\alpha) \geq (1 - \delta) J^{\pi^M}(\alpha). \tag{8}$$

Proposition 3 *Suppose that* $J^{\pi^M}(\alpha) \neq 0$ *and define*

$$\tilde{v}_n(s) = \left|v^M(s) - v_n(s)\right| \quad \forall s \in S, n \in \mathbb{Z}_+.$$

Let $\{\sigma_n\}_{n\geq 1}$ *be a convergent sequence of state ordering rules which induces a sequence of nonnegative numbers* $\{\delta_n\}_{n\geq 1}$ *satisfying* $\delta_n \to 0^+$ *and* $\delta_n \geq \tilde{v}_n(s)$ *for all* $s \in \mathcal{S}, n \in \mathbb{Z}_+$. *Then, there exists a positive integer* $N < \infty$ *such that Algorithm 1 will converge after* N *iterations to a* $\left(\frac{\delta_N}{J^{\pi^M}(\alpha)}\right)$*-optimal monotone policy.*

Proof First, we show that MPI converges. For all $s \in S$ and $n \geq 0$, we have

$$\begin{aligned} 0 \leq \left|v_n(s) - v_{n-1}(s)\right| =& \left|v_n(s) - v^M(s) + v^M(s) - v_{n-1}(s)\right| \\ \leq& \left|v_n(s) - v^M(s)\right| + \left|v^M(s) - v_{n-1}(s)\right| \quad (9) \\ \leq& \delta_n + \delta_{n-1}, \quad (10) \end{aligned}$$

where (9) uses the triangle inequality and (10) uses the definition of $\tilde{v}_n(s)$ along with the fact that $\delta_n \geq \tilde{v}_n(s)$ for all $s \in \mathcal{S}$. Since $\delta_n \to 0^+$, we have $\delta_n + \delta_{n-1} \to 0^+$, and by the sandwich theorem, we have $|v_n(s) - v_{n-1}(s)| \to 0^+$. Since this holds for all $s \in \mathcal{S}$, the positivity properties of norms imply that $\|\boldsymbol{v}_n - \boldsymbol{v}_{n-1}\| \to 0^+$. Thus, by definition of convergence, for every ϵ we can find an $N \geq 1$ such that for every $n \geq N$, $\|\boldsymbol{v}_n - \boldsymbol{v}_{n-1}\| < \epsilon$, and the termination criteria in the MPI algorithm is satisfied.

Next, we show that the resulting policy π_N, with associated value function $\boldsymbol{v}_N$, is a δ_N-optimal monotone policy. We have

$$\begin{aligned} J^{\pi^M}(\alpha) - J^{\pi}(\alpha) &= \sum_{s\in\mathcal{S}} \alpha(s)(v^M(s) - v_N(s)) \\ &\leq \sum_{s\in\mathcal{S}} \alpha(s)|v^M(s) - v_N(s)| \\ &\leq \sum_{s\in\mathcal{S}} \alpha(s)\delta_N = \delta_N. \end{aligned}$$

Re-arranging terms yields

$$J^{\pi_N}(\alpha) \geq \left(1 - \frac{\delta_N}{J^{\pi^M}(\alpha)}\right) J^{\pi^M}(\alpha).$$

Thus, π_N is a $\left(\frac{\delta_N}{J^{\pi^M}(\alpha)}\right)$-optimal monotone policy. □

We now identify conditions on the existence of a state ordering rule such that MPI converges to an optimal monotone policy. We remark that Lemma 4 is similar to Lemma 3 in Mansour and Singh (1999).

Lemma 4 *Let* π *and* $\bar{\pi}$ *be two policies (with corresponding value functions* $\boldsymbol{v}^{\pi}$ *and* $\boldsymbol{v}^{\bar{\pi}}$*, respectively) whose actions differ in only one state* $\bar{s}$*, i.e.,* $\pi(s) = \bar{\pi}(s)$ *for* $s \neq \bar{s} \in \mathcal{S}$. *Suppose that*

$$v^{\pi}(\bar{s}) = r(\bar{s}, \pi(\bar{s})) + \gamma \sum_{s'\in\mathcal{S}} P(s'|\bar{s}, \pi(\bar{s}))v^{\pi}(s') < r(\bar{s}, \bar{\pi}(\bar{s})) + \gamma \sum_{s'\in\mathcal{S}} P(s'|\bar{s}, \bar{\pi}(\bar{s}))v^{\pi}(s').$$

Then, $v^{\pi}(s) < v^{\bar{\pi}}(s)$ *for all* $s \in \mathcal{S}$.

Proof For $s \neq \bar{s}$, we can continually expand v^{π} to get

$$
\begin{aligned}
v^{\pi}(s) &= r(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,\pi(s)) v^{\pi}(s') && (11)\\
&= r(s,\bar{\pi}(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,\bar{\pi}(s)) v^{\pi}(s') && (12)\\
&= r(s,\bar{\pi}(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,\bar{\pi}(s)) \big(r(s',\pi(s')) + \gamma \sum_{s'' \in \mathcal{S}} P(s''|s',\pi(s')) v^{\pi}(s'')\big)\\
&< r(s,\bar{\pi}(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,\bar{\pi}(s)) \big(r(s',\bar{\pi}(s')) + \gamma \sum_{s'' \in \mathcal{S}} P(s''|s',\bar{\pi}(s')) v^{\pi}(s'')\big)\\
&\vdots\\
&= v^{\bar{\pi}}(s).
\end{aligned}
$$

The same reasoning applies for $\bar{s}$ with strict inequality in (12). Thus, $v^{\pi}(s) < v^{\bar{\pi}}(s)$ for all $s \in \mathcal{S}$. □

Lemma 5 *Suppose we have an optimal monotone policy π^M with corresponding value function v^M. For any $s \in \mathcal{S}$ and $a \in \mathcal{A}$, define the policy*

$$
\pi'_{s,a}(s') := \begin{cases} \pi^M(s'), & s' \neq s \\ a, & s' = s, \end{cases}
$$

and let the set $\mathcal{A}'(s) = \left\{a \in \mathcal{A} : \pi'_{s,a} \in \Pi^M\right\}$, i.e., the set of all actions where $\pi'_{s,a}$ is monotone. Then, π^M satisfies

$$
\pi^M(s) \in \arg\max_{a \in \mathcal{A}'(s)} \left\{ r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v^M(s') \right\}, \tag{13}
$$

for all $s \in \mathcal{S}$.

Proof First, note that since π^M is an optimal monotone policy, for all $\pi \in \Pi^M$ with corresponding value function $\boldsymbol{v}^{\pi}$, we have

$$
\begin{aligned}
& J^{\pi^M}(\alpha) \geq J^{\pi}(\alpha) \\
\Leftrightarrow & \sum_{s \in \mathcal{S}} \alpha(s) v^M(s) \geq \sum_{s \in \mathcal{S}} \alpha(s) v^{\pi}(s).
\end{aligned}
$$

This implies that for any $\pi \in \Pi^M$, there exists at least one $s \in \mathcal{S}$ for which $v^M(s) \geq v^{\pi}(s)$.

Now suppose for the purpose of contradiction that π^M is an optimal monotone policy but there exists an $\bar{s} \in \mathcal{S}$ for which $\pi^M(\bar{s}) \notin \arg\max_{a \in \mathcal{A}'(\bar{s})}\{r(\bar{s},a) + \gamma \sum_{s' \in S} P(s'|\bar{s},a) v^M(s')\}$. Then, there must exist an $a' \in \mathcal{A}'(\bar{s})$ such that

$$
r(\bar{s},\pi^M(\bar{s})) + \gamma \sum_{s' \in S} P(s'|\bar{s},\pi^M(\bar{s})) v^M(s') < r(\bar{s},a') + \gamma \sum_{s' \in S} P(s'|\bar{s},a') v^M(s').
$$

By Lemma 4, $v^M(s) < v^{\pi'_{s,a'}}(s)$ for all $s \in \mathcal{S}$, which is a contradiction. □

Proposition 6 *Suppose we have an input vector $\boldsymbol{v}_0$ to Algorithm 1 which satisfies*

$$\boldsymbol{v}_0 \in \Bigg\{ \boldsymbol{v} \in \mathbb{R}^S : \underset{a \in \mathcal{A}(s)}{\arg\max}\, r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v^M(s) = \underset{a \in \mathcal{A}(s)}{\arg\max}\, r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v(s') \text{ for all } s \in \mathcal{S} \Bigg\}, \tag{14}$$

i.e., the input vector $\boldsymbol{v}_0$ yields the same actions as an optimal monotone value function $\boldsymbol{v}^M$. Then, under a state ordering rule which guarantees that $\mathcal{A}_n(s) = \mathcal{A}'(s)$ for all $s = \sigma_n(i), i = 1, \ldots, S$ for $n \in \{1, 2\}$, the algorithm will converge to a corresponding optimal monotone policy π^M in a maximum of two iterations.

Proof For all $s = \sigma_n(i), i = 1, \ldots, S$ for $n \in \{1, 2\}$, the algorithm finds $\pi_n(s) = \pi^M(s)$ by Lemma 5. Thus, $\boldsymbol{v}_1 = \boldsymbol{v}_2 = \boldsymbol{v}^M$. If $\|\boldsymbol{v}_1 - \boldsymbol{v}_0\| \leq \epsilon$, we are done. Otherwise, the algorithm terminates after two iterations for arbitrary ϵ, since $\|\boldsymbol{v}_2 - \boldsymbol{v}_1\| = 0$. □

Remark A similar result to Proposition 6 could be derived by assuming that $\mathcal{A}_n(s) \subseteq \mathcal{A}'(s)$. However, MPI is designed such that $\mathcal{A}_n(s) = \mathcal{A}'(s)$ and from numerical testing, we found that typically, $\mathcal{A}_n(s) \supset \mathcal{A}'(s)$.

Proposition 6 provides sufficient conditions under which MPI converges to an optimal monotone policy π^M after two iterations. Accordingly, if MPI ever reaches an optimal monotone policy, it will necessarily terminate within two iterations if these conditions are met. While these conditions may be challenging to verify in practice, we note that the result highlights the importance of the state ordering rule and its interaction with the initial vector $\boldsymbol{v}_0$. Specifically, it is not enough to have an initial vector which guarantees the preservation of the actions in an optimal monotone policy — the state ordering rule must also guarantee that the actions corresponding to π^M are not excluded from $\mathcal{A}_n(s)$ for some $s \in \mathcal{S}$ during the MPI algorithm. To this end, we show in the following result that such a state ordering rule can be derived for an MDP with two actions.

Corollary 6.1 *Suppose we have an MDP with $A = 2$. Given an optimal monotone value function $\boldsymbol{v}^M$ as an input $\boldsymbol{v}_0$ to MPI, there exists a state ordering rule such that the algorithm converges to π^M in one iteration.*

Proof Because there are only two actions in $\mathcal{A}$, there are three cases for the structure of an optimal monotone policy $\pi^M(s)$:

(i) There exists $\bar{s}$ such that $\pi^M(s) = 1$ for $s < \bar{s}$ and $\pi^M(s) = 2$ for $s \geq \bar{s}$.

(ii) $\pi^M(s) = 1$ for all $s \in \mathcal{S}$.

(iii) $\pi^M(s) = 2$ for all $s \in \mathcal{S}$.

First, we give a state ordering rule for each case which guarantees that $\mathcal{A}_1(s) \subseteq \mathcal{A}'(s)$ for all $s = \sigma_1(i), i = 1, \ldots, S$.

In case (i), let $\sigma_1(1) = \bar{s}$ and $\sigma_1(2) = \bar{s} - 1$. In the first iteration of the algorithm for $i = 1$, $\mathcal{S}^- = \mathcal{S}^+ = \emptyset$, so $A_{\min}(\bar{s}) = 1$, $A_{\max}(\bar{s}) = 2$, and $A_1(\bar{s}) = \{a \in \mathcal{A} : 1 \leq a \leq 2\} = \mathcal{A}$. We have $\pi'_{\bar{s},1} \in \Pi^M$ since $\pi^M(\bar{s} - 1) = 1$ and $\pi^M(\bar{s} + 1) = 2$, so letting $\pi'_{\bar{s},1}(\bar{s}) = 1$ results in a policy which is still monotone. Trivially, $\pi^M = \pi'_{\bar{s},2} \in \Pi^M$. Then $\mathcal{A}'(\bar{s}) = \{1, 2\} = \mathcal{A}$ and $A_1(\bar{s}) \subseteq \mathcal{A}'(\bar{s})$. Similarly, $A_1(\bar{s} - 1) \subseteq \mathcal{A}'(\bar{s} - 1)$. For $i > 2$, if $\sigma_1(i) < \bar{s} - 1$, we have $A_1(\sigma_1(i)) = 1$ and $\mathcal{A}'(\sigma_1(i)) = 1$, and if $\sigma_1(i) > \bar{s}$, we have $A_1(\sigma_1(i)) = 2$ and $\mathcal{A}'(\sigma_1(i)) = 2$. Thus, $A_1(s) \subseteq \mathcal{A}'(s)$ for all $s = \sigma_1(i), i = 1, \ldots, S$.

In case (ii), let $\sigma_1(1) = S$, and in case (iii), let $\sigma_1(1) = 1$; in both cases, the rest of the states can be visited in arbitrary order. The proofs for both cases are very similar to that of case (i) and are thus omitted.

Then, since $\boldsymbol{v}^M$ clearly satisfies equation (14), we can apply Proposition 6 for the desired result. Because $\boldsymbol{v}^0 = \boldsymbol{v}^M$, the algorithm terminates after one iteration for arbitrary ϵ. □

In summary, our analysis of the MPI algorithm shows that although there exist conditions under which the algorithm converges to an optimal or near-optimal monotone policy, these conditions may be hard to verify. Furthermore, even when the algorithm is initialized with a monotone optimal value function, a state ordering rule which satisfies certain conditions must be used to guarantee that a monotone optimal policy is returned. In general, MPI may not converge, and may not converge to an optimal monotone policy. In the next section, we introduce the MMPI algorithm, which addresses these shortcomings.

2.4 Modified Monotone Policy Iteration

In this section, we propose the MMPI algorithm (Algorithm 2) and provide an analysis of its convergence as well as the quality of the returned policy in comparison to that of the MPI algorithm (Algorithm 1). MMPI uses the same policy improvement step as MPI (described in §2.2); however, in MMPI, we terminate when the algorithm revisits a policy that has been seen before instead of terminating based on the distance between subsequent value functions as in MPI. Furthermore, MMPI keeps track of the best policy seen by the algorithm (i.e., the policy yielding the highest expected total discounted reward) and returns the best policy seen at termination. Note that unlike MPI, MMPI takes into account the initial state probabilities α. In Theorem 7, we show that in contrast to MPI, the MMPI algorithm is guaranteed to converge in a finite number of iterations.

Theorem 7 *The MMPI algorithm converges in a maximum of* $\binom{S+A+1}{A-1} + 1$ *iterations.*

Algorithm 2 Modified Monotone Policy Iteration

Require: $\{\sigma_n\}_{n\geq 1}$, $\boldsymbol{v}_0$, $n = 0$, α
Ensure: $\sigma_n : \{1, ..., S\} \to \{1, ..., S\}$ is a one-to-one mapping for each n
$V \Leftarrow \emptyset$, $J_{best} \Leftarrow -\infty$
do
  if $n > 0$ **then**
    $V \Leftarrow V \cup \{\pi_n\}$
  end if
  $n \Leftarrow n + 1$
  $\bar{\mathcal{S}} \Leftarrow \emptyset$
  for $i = 1$ to S **do**
    $s \Leftarrow \sigma_n(i)$
    $\mathcal{S}^- \Leftarrow \{s' \in \bar{\mathcal{S}} : s' < s\}$
    $\mathcal{S}^+ \Leftarrow \{s' \in \bar{\mathcal{S}} : s' > s\}$
    if $\mathcal{S}^- = \emptyset$ **then**
      $A_{\min}(s) \Leftarrow 1$
    else
      $A_{\min}(s) \Leftarrow \pi_n(\max \mathcal{S}^-)$
    end if
    if $\mathcal{S}^+ = \emptyset$ **then**
      $A_{\max}(s) \Leftarrow A$
    else
      $A_{\max}(s) \Leftarrow \pi_n(\min \mathcal{S}^+)$
    end if
    $\mathcal{A}_n(s) \Leftarrow \{a \in \mathcal{A} : A_{\min}(s) \leq a \leq A_{\max}(s)\}$
    $\pi_n(s) \Leftarrow \arg\max_{a \in \mathcal{A}_n(s)} r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v_{n-1}(s')$
    $\bar{\mathcal{S}} \Leftarrow \bar{\mathcal{S}} \cup \{s\}$
  end for
  Perform policy evaluation on π_n to obtain $\boldsymbol{v}_n$
  $J_n = \sum_{s \in \mathcal{S}} \alpha(s) v_n(s)$
  if $J_n > J_{best}$ **then**
    $\pi_{best} = \pi_n$
    $J_{best} = J_n$
  end if
while $\pi_n \notin V$
return π_{best}

Proof The algorithm terminates in iteration n if π_n has been found by the algorithm in a previous iteration. We can verify from the policy improvement step that the algorithm is restricted to finding monotone policies; thus, the algorithm can find a maximum of $\binom{S+A+1}{A-1}1$ (Garcia et al, 2022) unique monotone policies before revisiting a monotone policy. Then, the maximum number of iterations is $\binom{S+A+1}{A-1} + 1$. □

We remark that as MMPI uses the same policy improvement procedure as MPI, some optimality results from the previous section – namely, Proposition 6 and Corollary 6.1 – are still valid for the MMPI algorithm. In the following result, we show that MMPI yields a better policy than MPI under certain conditions. We first define a cyclic state ordering rule.

Definition 3 (Cyclic State Ordering Rule) A state ordering rule $\{\sigma_n\}_{n\geq 1}$ is called *cyclic* with cycle-length k if for any integer $n \geq 1$, there exists an integer $k \geq 1$ such that $\pi_n = \pi_{n+k}$ implies that $\pi_{n+m} = \pi_{n+k+m}$ for any integer $m \geq 1$ under the MPI algorithm.

Proposition 8 *Let π_n^1 and π_n^2 denote the policies found in iteration n of MPI and MMPI respectively. Let N_1 denote the number of iterations after which MPI terminates if it does; if not, let $N_1 = \infty$. Similarly, let N_2 denote the number of iterations after which MMPI terminates, which is guaranteed to be finite (Theorem 7). Let $\hat{\pi}^1$ and $\hat{\pi}^2$ denote the policy returned by MPI (if one exists) and MMPI respectively.*

(a) Suppose we carry out the MPI and MMPI algorithms with the same inputs $\{\sigma_n\}_{n\geq 1}$ and $\boldsymbol{v}_0$. If $N_1 \leq N_2$, then $J^{\hat{\pi}^1}(\alpha) \leq J^{\hat{\pi}^2}(\alpha)$.

(b) Suppose we carry out the MPI and MMPI algorithms with the same input $\boldsymbol{v}_0$ and cyclic state-ordering rule $\{\sigma_n\}_{n\geq 1}$. Define

$$\hat{\pi}_n^1 = \begin{cases} \pi_n^1 & \text{if } n \leq N_1 \\ \hat{\pi}^1 & \text{if } n > N_1. \end{cases}$$

Similarly, define

$$\hat{\pi}_n^2 = \begin{cases} \pi_{best_n}^2 & \text{if } n \leq N_2 \\ \hat{\pi}^2 & \text{if } n > N_2, \end{cases}$$

where $\pi_{best_n}^2$ denotes the policy assigned to π_{best} in iteration n of MMPI. Then, $J^{\hat{\pi}_n^2}(\alpha) \geq J^{\hat{\pi}_n^1}(\alpha)$ for all $n \geq 1$.

Proof First, we prove part (a). Since we have the same inputs for both algorithms, the same state ordering rule $\sigma_{n\geq 1}$, and the same policy improvement procedure, we have $\hat{\pi}^1 = \pi_{N_1}^1 = \pi_{N_1}^2$. By design, $J^{\pi_n^2}(\alpha) \leq J^{\hat{\pi}^2}(\alpha)$ for all $n \leq N_2$. Since $N_1 \leq N_2$, $J^{\hat{\pi}^1}(\alpha) = J^{\pi_{N_1}^2}(\alpha) \leq J^{\hat{\pi}^2}(\alpha)$.

To prove part (b), we only show the case when $N_1 > N_2$ as the $N_1 \leq N_2$ case follows from similar logic as part (a). Then, when $n \leq N_2$, the same logic as part (a) gives that $\hat{\pi}_n^1 = \hat{\pi}_n^2$, so $J^{\hat{\pi}_n^2}(\alpha) \geq J^{\hat{\pi}_n^1}(\alpha)$. Now, if $N_2 < n \leq N_1$, there must exist an $n' < N_2$ such that $\hat{\pi}_{n'}^1 = \hat{\pi}_{n'}^2 = \hat{\pi}_{N_2}^1 = \hat{\pi}_{N_2}^2$ since MMPI terminated at N_2, and such that the cycle-length of the cyclic state ordering rule is $k = N_2 - n'$. We can write $n = N_2 + \hat{m}k + m$ where $\hat{m} \in \mathbb{Z}_{\geq 0}$ is some non-negative integer and $1 \leq m < k$. Then,

$$\hat{\pi}_n^1 = \hat{\pi}_{N_2+\hat{m}k+m}^1 = \hat{\pi}_{N_2+m}^1 = \hat{\pi}_{n'+m}^1 = \hat{\pi}_{n'+m}^2.$$

Now, $n' + m < n' + k = N_2$, so $J^{\hat{\pi}_n^2}(\alpha) \geq J^{\hat{\pi}_{n'+m}^2}(\alpha) = J^{\hat{\pi}_{n'+m}^1}(\alpha) = J^{\hat{\pi}_n^1}(\alpha)$. Finally, the proof for $n > N_1$ follows by letting $n = N_1$ in the proof for $N_2 < n \leq N_1$. □

Proposition 8 shows that, given the same input and the same cyclic state-ordering rule, MMPI returns a policy which has an expected total discounted reward that is at least as high as that of the policy returned by MPI. This holds even when MPI does not converge and the algorithm must be terminated by specifying a maximum number of iterations, for example. We remark that in the MMPI algorithm, we could keep track of value functions instead of policies, terminating when the algorithm revisits a value function seen before. This may accelerate the convergence of the algorithm when multiple policies map to the same value function; however, it may require more memory and introduce numerical precision issues. In our computational analyses (see §4 and §5), we design several state ordering rules for the MMPI algorithm and analyze their performance with regard to optimality and computational time.

3 Machine Maintenance with Stochastic Repairs

The problem of identifying an optimal monotone policy — even when the optimal policy is not monotone — is of great practical interest when the underlying MDP has significant structure suggesting that a monotone policy may perform well. Our goal in this section is to formulate a problem that exhibits significant problem structure guaranteeing the existence of an optimal monotone policy. Then, we ultimately plan to perturb the problem data in our numerical experiments (see §4) so that the optimal policy is no longer guaranteed to be monotone, but an optimal monotone policy may still be of practical interest.

Accordingly, we motivate our choice of studying machine maintenance as an optimization problem with significant structure in §3.1. Next, we formulate a scalable single-unit condition-based optimization problem with multiple stochastic repair options in §3.2. Then, we show in §3.3 that this formulation is guaranteed to have an optimal monotone policy. Our computational experiments (see §4) involve perturbations of this problem with varying sizes of $\mathcal{S}$ and $\mathcal{A}$.

3.1 Condition-based Maintenance Optimization

Maintenance optimization problems comprise a large class of well-studied problems with such structure (Osaka, 2002; Wang, 2002; de Jonge and Scarf, 2020). Specifically, in condition-based maintenance optimization, a decision-maker monitors the condition of one or more degrading systems over time and must determine if and when to perform maintenance actions on the system (Jardine et al, 2006). Notably, the degradation process is often assumed to be irreversible and assumed to worsen over time (Elwany et al, 2011). Maintenance actions often include replacement, which is very costly but resets the system's degradation state, and/or repair, which is less costly than replacement but improves the system's degradation state by a lesser degree. Due to the sequential nature of monitoring and repair decisions, condition-based maintenance problems and their variations are often modeled as an MDP (Kurt and Maillart, 2009; Kurt and Kharoufeh, 2010; Elwany et al, 2011; Ulukus et al, 2012;

Liu et al, 2017; Zheng and Makis, 2020; Zhang and Zhang, 2023). Moreover, for many variations of the problem considering only replacement as a repair option, it the optimal policy may exhibit a control-limit structure, i.e., replacement is optimal once the system's degradation state exceeds some threshold (Kurt and Maillart, 2009; Kurt and Kharoufeh, 2010; Ulukus et al, 2012; Liu et al, 2017). Likewise, variations of the problem which include some repair actions exhibit optimal monotone policies (Zheng and Makis, 2020).

3.2 Model Formulation

We now formulate a general single-unit condition-based optimization problem with machine replacement and multiple stochastic repair options. Let $s_t \in \mathcal{S} = \{0, 1, ..., S\}$ denote the condition of the system in time t, with each state representing increasing levels of degradation. After the decision-maker observes the system state, she selects a repair option $a_t \in \mathcal{A} = \{0, ..., A\}$. The repair actions $a = 0$ and $a = A$ correspond to no repair and machine replacement, respectively, whereas all other $a \in \mathcal{A} \setminus \{0, A\}$ correspond to stochastic repair actions of increasing quality. To be precise, each $a \in \mathcal{A}$ stochastically repairs the condition of the machine by Y_t^a, where the random variable Y_t^a has the following probability mass function:

$$\mathbb{P}(Y_t^a = y) = \begin{cases} 1 & a = 0, y = 0 \text{ or } a = A, y = S \\ 2y/(a(a+1)) & a \neq 0, A \text{ and } y = 0, ..., a \\ 0 & \text{otherwise.} \end{cases}$$

Note that since the machine's condition cannot be improved beyond 0, the machine's condition after repair is given by $Z_t = \max\{0, s_t - Y_t^a\}$. After repair, the machine operates and experiences X_t shocks, where each shock degrades the machine's condition by 1 unit. The random variable X_t is Geometrically distributed with parameter ρ, i.e., $\mathbb{P}(X_t = x) = (1-\rho)^x \rho$ for any non-negative integer $x \in \mathbb{Z}_+$. Since the machine cannot degrade beyond condition S, the post-deterioration state is given by $s_{t+1} = \min\{S, Z_t + X_t\}$. With these dynamics, we determine the transition probabilities by evaluating $P(s'|s, a) = P(s' = \min\{S, \max\{0, s - Y_t^a\} + X_t\})$. We illustrate these transition dynamics in Figure 2, and we model the transition probability matrix

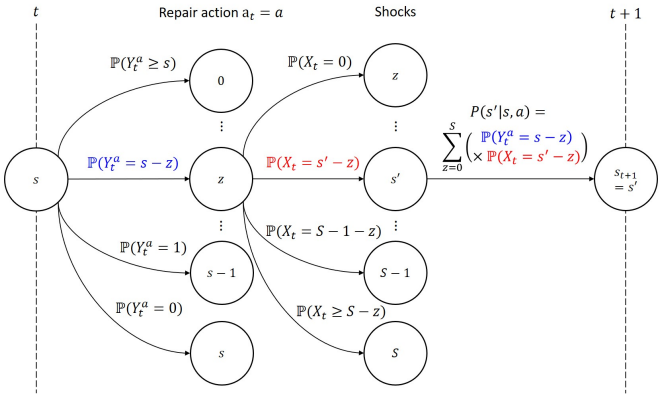

Figure 2 Illustration of transition dynamics and transition probabilities for the machine maintenance problem

with components

$$P(s'|s,a) = \begin{cases} \sum_{y=0}^{a} (1-\rho)^{s'-s+y} \rho \frac{2y}{a(a+1)} & a < A, a \leq s, s' < S \\ \sum_{y=0}^{a} (1-\rho)^{S-s+y} \frac{2y}{a(a+1)} & a < A, a \leq s, s' = S \\ \sum_{y=0}^{s-1} (1-\rho)^{s'-s+y} \rho \frac{2y}{a(a+1)} + \sum_{y=s}^{a} (1-\rho)^{s'} \rho \frac{2y}{a(a+1)} & s \leq a < A, s' < S \\ \sum_{y=0}^{s-1} (1-\rho)^{S-s+y} \frac{2y}{a(a+1)} + \sum_{y=s}^{a} (1-\rho)^{S} \frac{2y}{a(a+1)} & s \leq a < A, s' = S \\ (1-\rho)^{s'} \rho & a = A, s' < S \\ (1-\rho)^{S} & a = A, s' = S \\ 0 & \text{otherwise.} \end{cases}$$

In every period, the decision-maker incurs a cost given by $r(s,a) = -(c_1(s) + c_2(a))$, where $c_1(s)$ is a fixed maintenance cost and $c_2(a)$ is a repair-dependent cost. We assume that $c_1(s)$ is increasing in s and $c_2(a)$ is increasing in a. The decision-maker's objective is to maximize her expected total discounted reward, or equivalently, minimize her expected total discounted cost.This objective results in the following value function:

$$v^*(s) = \max_{a \in \mathcal{A}} \left\{ -(c_1(s) + c_2(a)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v(s') \right\} \text{ for all } s \in \mathcal{S}.$$

An optimal policy $\pi^* \in \Pi$ identifies a cost-minimizing action in each state. In the next section, we show that there exists an optimal policy π^* that is monotone.

3.3 Analytical Properties

We now show that there is guaranteed to be an optimal policy that is monotone. That is, there exists $\pi^* \in \Pi^M$. To prove this result, we begin with the following definitions.

Definition 4 (First-order Stochastic Dominance (Krishnamurthy, 2016)) Let f_1 and f_2 be any two probability mass functions on $\mathcal{S}$. The distribution f_1 has first-order stochastic dominance on f_2 (denoted by $f_1 \succeq f_2$) if $\sum_{k \geq s} f_1(k) \geq \sum_{k \geq s} f_2(k)$ for any $s \in \mathcal{S}$.

Definition 5 (Supermodular) A function $g : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is supermodular on $\mathcal{S} \times \mathcal{A}$ if $g(s^+, a^+) + g(s^-, a^-) \geq g(s^+, a^-) + g(s^-, a^+)$ for any $s^+, s^- \in \mathcal{S}$ with $s^+ \geq s^-$ and $a^+, a^- \in \mathcal{A}$ with $a^+ \geq a^-$.

These definitions are standard tools in monotone comparative statistics. We now use these tools to highlight key properties of our machine maintenance problem.

Proposition 9 *The following properties hold.*

1. $P(\cdot|s+1, a) \succeq P(\cdot|s, a)$ *for any* $s \leq S-1$, $a \in \mathcal{A}$.
2. $P(\cdot|s, a) \succeq P(\cdot|s, a+1)$ *for any* $a \leq A-1$, $s \in \mathcal{S}$.
3. *The optimal value function* $v^*(s)$ *is non-increasing in* s.

Proof The proof for each property proceeds as follows.

1. It suffices to show that $\sum_{k \geq s'} P(k|s+1, a) \geq P(k|s, a)$ for any $k \in \mathcal{S}$, $s \leq S-1$, and $a \in \mathcal{A}$. Equivalently, we must show that $\mathbb{P}(s_{t+1} \geq s'|s_t = s, a_t = a)$ is increasing in s. Take any $s \in \mathcal{S}$, $s \leq S-1$, and consider each of the following cases.
   - **Case 1:** Suppose that $a \in \mathcal{A}$, $a \leq A-1$. We have
   $$\begin{aligned}
   &\mathbb{P}(s_{t+1} \geq s'|s_t = s, a_t = a) = \mathbb{P}(Z_t + X_t = s'|s, a) \\
   &= \sum_{z=0}^{s'} \mathbb{P}(X_t \geq s' - z|Z_t = z, s, a)\mathbb{P}(Z_t = z|s, a) \\
   &= \sum_{z=0}^{s'} \mathbb{P}(X_t \geq s' - z)\mathbb{P}(\max\{0, s - Y_t^a\} = z|s, a) \\
   &= \sum_{z=0}^{s'} (1-\rho)^{s'-z+1} \left(\frac{2(s-z)}{a(a+1)}\right) + (1-\rho)^{s'+1} \sum_{y=s+1}^{a} \frac{2y}{a(a+1)}. \qquad (15)
   \end{aligned}$$
   Notice that the expression in (15) is increasing in s. Thus, $P(\cdot|s+1, a) \succeq P(\cdot|s, a)$ for any $s \leq S-1$, $1 \leq a \leq A-1$.
   - **Case 2:** For the case where $a = A$, we have $\mathbb{P}(Z_t = 0) = 1$. Therefore,
   $$\mathbb{P}(s_{t+1} \geq s'|s_t = s, a_t = A) = \mathbb{P}(X_t \geq s') = (1-\rho)^{s'+1},$$
   which is equal for all $s \in \mathcal{S}$. Thus, $P(\cdot|s+1, A) \succeq P(\cdot|s, A)$ for any $s \leq S-1$.

- **Case 3:** Finally, suppose that $a = 0$, we have $\mathbb{P}(Z_t = s) = 1$. Therefore,

$$\mathbb{P}(s_{t+1} \geq s'|s_t = s, a_t = 0) = \mathbb{P}(X_t \geq \max\{0, s' - s\}) = \begin{cases} 1 & s' \leq s \\ (1-\rho)^{s'-s+1} & s' > s \end{cases}.$$

  Clearly, this expression is increasing in s. Thus, $P(\cdot|s+1, a) \succeq P(\cdot|s, a)$ for any $s \leq S-1$, $a = 0$.

Since $P(\cdot|s+1, a) \succeq P(\cdot|s, a)$ for all $s \leq S-1$ and $a \in \mathcal{A}$, the proof is complete.

2. It suffices to show that $\mathbb{P}(s_{t+1} \geq s'|s_t = s, a_t = a)$ is non-increasing in a for any $s \in \mathcal{S}$ and $a \in \mathcal{A}$. Take any $s \in \mathcal{S}$ and consider the following cases.
   - **Case 1:** Suppose that $1 \leq a \leq A-2$. Since (15) is decreasing in a, $P(\cdot|s, a) \succeq P(\cdot|s, a+1)$ for any $1 \leq a \leq A-2$.
   - **Case 2:** Suppose that $a = A-1$. Then, we have that

$$\begin{aligned} &\mathbb{P}(s_{t+1} \geq s'|s_t = s, a_t = A) = \mathbb{P}(X_t \geq s') = (1-\rho)^{s'+1} \\ &= (1-\rho)^{s'+1}\left(\sum_{z=0}^{s'}\left(\frac{2(s-z)}{A(A-1)}\right) + \sum_{y=s+1}^{A-1}\frac{2y}{A(A-1)}\right) \\ &\leq \sum_{z=0}^{s'}(1-\rho)^{s'-z+1}\left(\frac{2(s-z)}{A(A-1)}\right) + (1-\rho)^{s'+1}\sum_{y=s+1}^{A-1}\frac{2y}{A(A-1)} \\ &= \mathbb{P}(s_{t+1} \geq s'|s_t = s, a_t = A-1). \end{aligned}$$

     Thus, $P(\cdot|s, A-1) \succeq P(\cdot|s, A)$.
   - **Case 3:** Suppose that $a = 0$. Then, we have

$$\begin{aligned} &\mathbb{P}(s_{t+1} \geq s'|s_t = s, a_t = 0) \\ &= \mathbb{P}(X_t \geq \max\{0, s' - s\}) = \begin{cases} 1 & s' \leq s \\ (1-\rho)^{s'-s+1} & s' > s \end{cases}. \end{aligned}$$

     If $s' \leq s$, then we have our desired result. Otherwise, if $s' > s$, then notice that

$$\begin{aligned} &\mathbb{P}(s_{t+1} \geq s'|s_t = s, a_t = 1) = \mathbb{P}(X_t \geq s' - s + 1|s_t = s, a_t = 1) \\ &= (1-\rho)^{s'-s+2} \leq (1-\rho)^{s'-s+1} = \mathbb{P}(s_{t+1} \geq s'|s_t = s, a_t = 0). \end{aligned}$$

     Thus, $P(\cdot|s, 0) \succeq P(\cdot|s, 1)$.

   Since $P(\cdot|s+1, a) \succeq P(\cdot|s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, the proof is complete.

3. Using induction on the value iteration algorithm, we show that $v^*(s)$ is non-increasing in s. Let $v_n(s)$ denote the value function for iteration n and let $\pi_n(s)$ denote the optimal action in iteration n, with $v_0(s) = 0$ for all $s \in \mathcal{S}$. By construction, the base case holds. Now, assume that $v_n(s)$ is non-increasing in s for $n = 0, 1, ..., t$. For $n = t+1$, we have

$$\begin{aligned} v_{t+1}(s+1) &= \max_{a \in \mathcal{A}}\left\{r(s+1, a) + \gamma\sum_{s' \in \mathcal{S}} P(s'|s+1, a)v_t(s')\right\} \\ &= r(s+1, \pi_{t+1}(s+1)) + \gamma\sum_{s' \in \mathcal{S}} P(s'|s+1, \pi_{t+1}(s+1))v_t(s') \end{aligned}$$

$$\leq r(s, \pi_{t+1}(s+1)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi_{t+1}(s+1)) v_t(s') \tag{16}$$

$$\leq r(s, \pi_{t+1}(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi_{t+1}(s)) v_t(s') \tag{17}$$

$$= v_{t+1}(s).$$

In the preceding equations, inequality (16) follows from the fact that $r(s,a)$ is non-increasing in s, the fact that $P(\cdot|s+1,a) \succeq P(\cdot|s,a)$, and the induction hypothesis (i.e. $v_t(s')$ is non-increasing in s'). Inequality (17) follows from the definition of $\pi_{t+1}(s)$. Thus, $v_n(s)$ is non-increasing in s for all n. Since $\lim_{n\to\infty} v_n(s) = v^*(s)$, we have our desired result.

□

Proposition 9 highlights several important properties of our machine maintenance problem. The first property implies that machines in worse condition are more likely to be in a worse future condition given the same repair action. The second property implies that machines receiving a higher quality repair action are more likely to be in better future condition than if the same machine were to receive a lower quality repair action. Finally, the third property implies that the cost incurred by performing the optimal action in each state increases as the machine's degradation increases. Altogether, these properties suggest that machines in worse condition should receive higher quality repair actions compared to machines in better condition. That is, the optimal policy should be monotone. Indeed, we prove this property in the following result.

Theorem 10 *There exists an optimal policy π^* for the machine maintenance problem that is monotone. That is, $\pi^*(s) \leq \pi^*(s')$ for any $s \leq s'$.*

Proof Topkis' Theorem (Topkis, 1978) states that there exists an optimal policy π^* that is monotone if the function $Q(s,a) := r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v^*(s')$ is supermodular on $\mathcal{S} \times \mathcal{A}$. Thus, it suffices to show that $Q(s,a)$ is supermodular on $\mathcal{S} \times \mathcal{A}$. To show this result, we first notice that by expanding terms, we have

$$\begin{aligned}
& Q(s+1,a+1) + Q(s,a) \geq Q(s+1,a) + Q(s,a+1) \\
\iff & -c_1(s+1) - c_1(s) - c_2(a+1) - c_2(a) \\
& \quad \gamma + \sum_{s' \in \mathcal{S}} \Big(P(s'|s+1,a+1) + P(s'|s,a) \Big) v^*(s') \\
& \geq -c_1(s+1) - c_1(s) - c_2(a+1) - c_2(a) \\
& \quad + \gamma \sum_{s' \in \mathcal{S}} \Big(P(s'|s+1,a) + P(s'|s,a+1) \Big) v^*(s') \\
\iff & \sum_{s' \in \mathcal{S}} \Big(P(s'|s+1,a+1) + P(s'|s,a) \Big) v^*(s') \\
& \quad \geq \sum_{s' \in \mathcal{S}} \Big(P(s'|s+1,a) + P(s'|s,a+1) \Big) v^*(s').
\end{aligned} \tag{18}$$

Thus, it suffices to show that (18) holds. Since $P(\cdot|s+1,a) \succeq P(\cdot|s,a)$ and $v^*(s')$ is non-increasing in s', we have

$$\sum_{s' \in \mathcal{S}} P(s'|s,a)v^*(s') \geq \sum_{s' \in \mathcal{S}} P(s'|s+1,a)v^*(s'). \tag{19}$$

Similarly, since $P(\cdot|s+1,a) \succeq P(\cdot|s+1,a+1)$ and $v^*(s')$ is non-decreasing in s', we have

$$\sum_{s' \in \mathcal{S}} P(s'|s+1,a+1)v^*(s') \geq \sum_{s' \in \mathcal{S}} P(s'|s+1,a)v^*(s'). \tag{20}$$

Adding (19) to (20) gives (18), which completes the proof. □

Since this machine maintenance problem has an optimal policy that is monotone, random perturbations of the transition probabilities could result in a machine maintenance problem whose optimal policy is monotone or almost monotone. As such, we believe that our machine maintenance problem is an ideal test bed for the numerical testing of Algorithm 1.

4 Computational Experiments - Machine Maintenance Problem

In this section, we assess the computational performance of various state orderings in the MMPI algorithm as applied to the machine maintenance problem described in §3.2 for varying sizes of $\mathcal{S}$ and $\mathcal{A}$. However, instead of implementing the problem exactly as described in §3.2, we perturbed the transition matrix P. Perturbing P from its nominal values could mean that a monotone policy is no longer optimal. Regardless, practitioners may still be interested in generating monotone policies for maintaining this machine due to underlying problem structure.

4.1 Implementation of the Machine Maintenance Problem

In our numerical implementation of the machine maintenance problem, we set $r_1(s) = 10s$ and $r_2(a) = 5a^2$. For each state space size $|\mathcal{S}| \in \{50, 100, 200, 400\}$ and action space size $|\mathcal{A}| \in \{20, 40\}$, we generated 40 instances of the machine maintenance problem with randomly perturbed transition probabilities and randomly generated initial distributions. To perturb the transition probabilities, we changed the distribution associated with Y_t^a and X_t. For $a \in \{1, ..., A-1\}$, we generated the non-zero probabilities $\left[\mathbb{P}(Y^a = a)\ \mathbb{P}(Y^a = a-1)\ ...\ \mathbb{P}(Y^a = 1)\right]$ according to a Dirichlet distribution with parameters $(a, a-1, ..., 0)$. We also randomly generated the parameter ρ_s associated with X_t for each $s \in \mathcal{S}$ according to a uniform distribution with support $[0.4, 0.6]$. We set the initial distribution α to be a vector with entries $\alpha(s) = 2(S-s)/(S(S+1))$. For each instance, we first found the optimal (non-monotone) policy using linear programming and recorded the optimal value functions $v^*(s)$. Then, we carried out MMPI initialized with the optimal value

functions using different state ordering rules σ, as described in the next subsection. The monotone policy resulting from the MMPI algorithm was then supplied as an initial feasible solution to formulation (6).

4.2 State Ordering Rules

In this section, we describe the state ordering rules σ considered in our numerical experiments. We investigated state ordering rules based on the following quantities, with the following motivations:

- $v_n(s)$: the state value function of state s at iteration n of the MMPI algorithm. A state ordering rule based on increasing (resp., decreasing) values of $v_n(s)$ prioritizes states that have the least (resp., greatest) value under policy π_n.
- $v^*(s) - v_n(s)$: the difference between the value of the optimal (non-monotone) policy and the state value function at iteration n of the MMPI algorithm. A state ordering rule based on increasing (resp., decreasing) quantities of $v^*(s) - v_n(s)$ prioritizes states that are closest to (resp., furthest from) the optimal state value under policy π_n.
- $\alpha(s)$: the initial probability that the machine is in state s. A state ordering rule based on increasing (resp., decreasing) values of $\alpha(s)$ prioritizes states that are most likely (resp., least likely) to be the initial state of the MDP.
- $\tau_n(s)$: the stationary distribution at state s of the Markov chain induced by the policy π_n. A state ordering rule based on increasing (resp., decreasing) values of $\tau_n(s)$ prioritizes states that are most likely (resp., least likely) to be occupied under policy π_n.

State ordering rules were constructed for each quantity by considering orderings from smallest to largest (i.e., increasing) and vice versa (i.e., decreasing). We also considered select two-factor interactions, i.e., the products of two quantities above, in both the increasing and decreasing cases. All state ordering rules are summarized in Table 1.

As a "control" state ordering rule, we included state ordering rule σ^0, which randomly permutes the set of states $\{1, \ldots, S\}$ at each iteration of Algorithm 1, and rule σ^1 (resp., σ^{10}), which orders the states by decreasing (resp., increasing) index. Note that rules σ^k, $k \in \{1, 4, 10, 13\}$, are rules whose state orderings do not change in each iteration of MMPI, while the rest are rules whose state orderings may change in each iteration of the algorithm.

4.3 Performance Metrics

We evaluated the performance of the MMPI algorithm for each state ordering rule under the following criteria:

- Optimality gap between the optimal monotone policy and the policy generated by the MMPI algorithm. We calculate this metric as $\left(J^{\pi'}(\alpha) - J^{\pi^{MMPI}}(\alpha)\right) \times 100/J^{\pi'}(\alpha)$ if the solver found an optimal solution to (6) within the time limit and $\pi' = \pi^*$ (i.e, the optimal MDP policy from solving (3)) otherwise. For every combination of $|\mathcal{S}|$ and $|\mathcal{A}|$,

Table 1 State ordering rules tested in the MMPI algorithm in computational experiments. In each entry of the table, the rule number corresponds to the state ordering rule that orders the states by decreasing (increasing) order of the product of the row title and column title. Rule 1 (10) orders the states by decreasing (increasing) index, and rule 0 (not shown in the table) corresponds to randomly shuffling the states.

	1	$v_n(s)$	$v^*(s) - v_n(s)$
1	σ^1 (σ^{10})	σ^2 (σ^{11})	σ^3 (σ^{12})
$\alpha(s)$	σ^4 (σ^{13})	$\sigma^5(\sigma^{14})$	σ^6 (σ^{15})
$\tau_n(s)$	σ^7 (σ^{16})	σ^8 (σ^{17})	σ^9 (σ^{18})

we compute this metric for each instance and report the mean, median, and standard deviation across all instances.

- Computation time (seconds) of the MMPI algorithm.
- Computation time (seconds) of MILP formulation (6) when warm-started with the resulting policy from the MMPI algorithm. Any instances that did not solve formulation (6) within the time limit were noted as such.

All experiments were performed on a computing cluster with Intel Xeon Gold 6226R 2.7GHz machines using the commercial optimization solver Gurobi 9.5.2. The time limit set in Gurobi for each problem instance was 1800 seconds.

4.4 Results

Our results on the quality of the policies generated and the computation time taken by the MMPI algorithm using the various state ordering rules are summarized in Figures 3 and 4 for action spaces of size $|\mathcal{A}| = 20$ and $|\mathcal{A}| = 40$, respectively (see Appendix A for details). In each figure, we highlight the "efficient" state ordering rules on the Pareto frontier of mean computational time and mean optimality gap computed across all instances. We find that there exists a small trade-off between the optimality gap of the policies generated and the running time of the state ordering rules, but that the fastest state ordering rules are still able to produce policies with relatively low optimality gaps.

We observe that state ordering rule 0, which was an efficient state ordering rule across all state and action space sizes, consistently yielded a policy which was either optimal or very close to optimal (0.1% optimality gap) in instances where an optimal monotone policy was found by the solver within the time limit. However, rule 0 also consistently took the longest amount of time, with mean computation times ranging from approximately three to five times that of the fastest state ordering rule for $|\mathcal{A}| = 20$ and approximately 11-18 times for $|\mathcal{A}| = 40$. As state ordering rule 0 randomly shuffles the states, it is able to explore a larger policy space than other state ordering rules - thereby yielding a higher quality policy at the expense of a longer computation time.

We also observe that state ordering rules 3 and 12, which orders states by $v^*(s) - v_n(s)$ (the gap between the value of the optimal (possibly non-monotone) policy and the state value function at iteration n of the MMPI

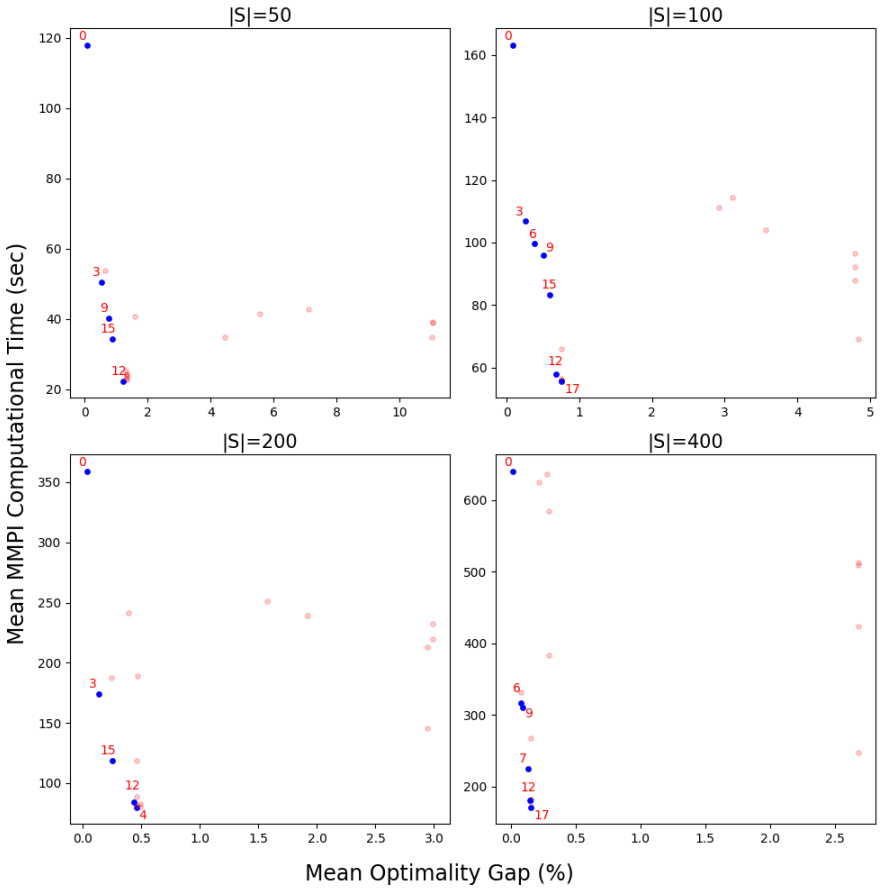

Figure 3 Trade-off curve representation of the mean computational time vs. mean optimality gap for various state ordering rules in the MMPI algorithm for machine maintenance MDPs, $|\mathcal{A}| = 20$. Blue dots (labeled) represent state ordering rules on the Pareto efficiency frontier. Red dots (unlabeled) represent state ordering rules that are dominated.

algorithm) in decreasing and increasing order, respectively, consistently appeared on or close to the Pareto frontier. Under the increasing ordering of rule 12, the MMPI algorithm terminated faster to a policy of worse quality than the decreasing ordering of rule 3. This may be due to the fact that rule 12 prioritizes states which are already close to optimal. Accordingly, from iteration to iteration, states that are already assigned optimal actions keep these actions while only states that are furthest from optimal may change their assigned actions. Similarly, state ordering rules 6 and 15, which order states by the product of $v^*(s) - v_n(s)$ and the initial probabilities $\alpha(s)$ in decreasing and increasing order respectively, also consistently appear on or near the Pareto frontier, with state ordering rule 15 producing policies with a higher optimality gap faster than state ordering rule 12. Across all state and action space sizes, the fastest state ordering rules produced policies with optimality

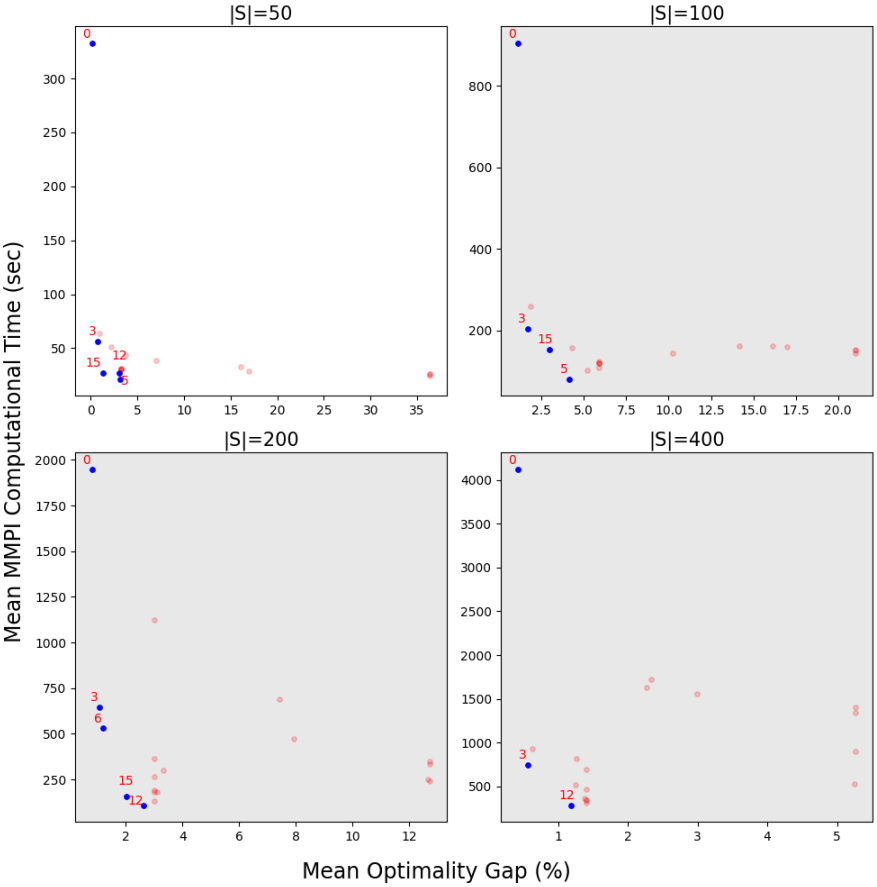

Figure 4 Trade-off curve representation of the mean computational time vs. mean optimality gap for various state ordering rules in the MMPI algorithm for machine maintenance MDPs, $|\mathcal{A}| = 40$. Blue dots (labeled) represent state ordering rules on the Pareto efficiency frontier. Red dots (unlabeled) represent state ordering rules that are dominated. State space(s) for which the monotone optimal policy is unknown for at least one instance (and thus for which the optimality gap was computed using the optimal policy upper bound) are indicated with a gray background.

gaps less than approximately 5%, suggesting that using these ordering rules in the MMPI algorithm, we can find monotone policies of high quality relatively quickly. While any state ordering in standard policy iteration will find an optimal monotone policy if the optimal policies are monotone, these state ordering rules may allow for faster convergence than others.

4.5 Computation Time of MILP Formulation with MMPI Warm-start

Figures 5 and 6 depict the total computation time for the solution procedure, i.e., the computation time for the MMPI algorithm to generate a warm start

policy which is then fed into MILP formulation (6), for $|\mathcal{A}| = 20$ and $|\mathcal{A}| = 40$ respectively (see Appendix A for a tabular summary).

Overall, the computation time of MILP formulation (6) for each state ordering rule was relatively consistent regardless of the quality of the warm starting solution from MMPI. For $|\mathcal{A}| = 20$, solving the MILP constituted the majority of the total computational time for the state ordering rules that completed the MMPI algorithm relatively quickly; for slower state ordering rules, completing the MMPI algorithm constituted the majority of the total computational time. Because Gurobi was not able to solve all of the instances for $|\mathcal{A}| = 40$ within the time limit (see Table 11 in Appendix A), it is difficult to generalize these results to higher action space sizes.

For $|\mathcal{S}| = 50$, we found that the MMPI algorithm scaled better than the MILP formulation with increasing action space size given warm-starting policies of similar quality. State ordering rule 6, for example, had a small increase in computational time for the MMPI algorithm from 54 seconds on average for $|\mathcal{A}| = 20$ to 64 seconds on average for $|\mathcal{A}| = 40$, producing policies with average optimality gaps of 0.07% and 0.09% respectively. When warm-started with these policies, however, the solver took approximately three times longer to solve the MILP on average for $|\mathcal{A}| = 40$ than $|\mathcal{A}| = 20$. This underscores the utility of algorithms such as MMPI in solving for high quality monotone policies particularly for MDPs with large action sizes when MILP formulations are intractable. In general, we found that the computation time for MILP formulation (6) increases much faster with action space size than with state space size. For sequential decision-making problems with structure suggesting the near-optimality monotone policies (e.g., the machine maintenance problem studied in this section), our results suggest that certain state ordering rules can help achieve a policy with very low optimality gap and low running times compared to the complete solution procedure with the MILP. For example, the results for the fastest state ordering rules (e.g., rule 12) were much faster than the MILP and near-optimal rules (e.g., rule 0) produced high-quality policies with optimality gaps around 0.1%. Accordingly, for such well-performing ordering rules, the additional computation time for solving the MILP may not be worth the improvement in the quality of the policy, especially for large action sets. To this end, we next explore the performance of MMPI under these state ordering for the case where the MDP lacks the type of structure inherent in machine maintenance problems.

5 Computational Experiments - Random MDPs

In this section, we assess the computational performance of various state orderings in the MMPI algorithm on randomly generated MDP instances to verify the robustness of the results from the machine maintenance application in §4. In contrast to the machine maintenance setting, random MDPs have no inherent structure.

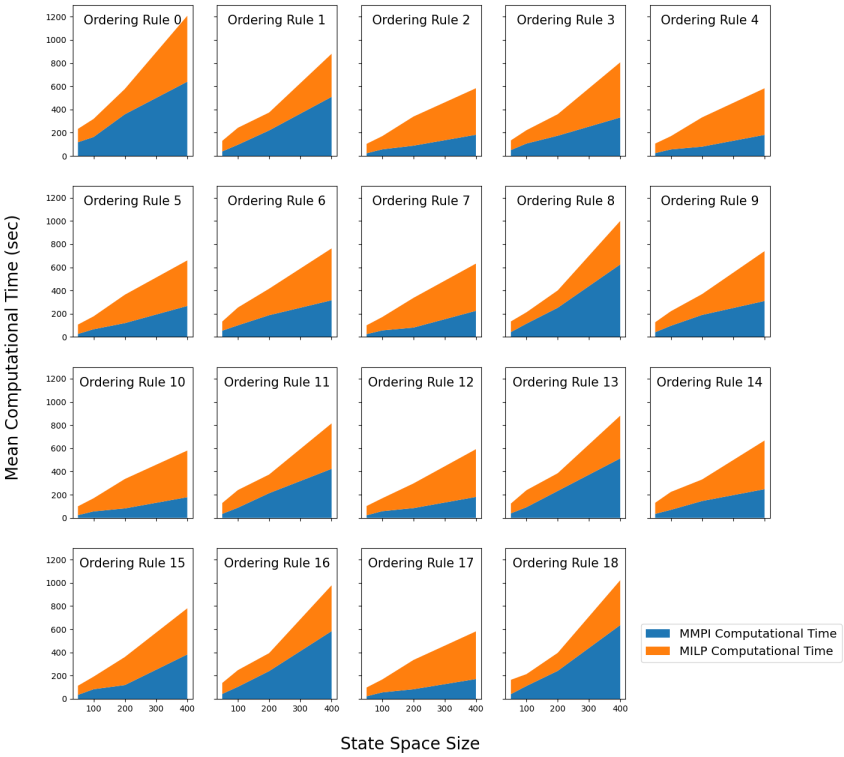

Figure 5 Total computation time for complete solution procedure (MMPI and MILP formulation (6)) for machine maintenance MDPs, $|\mathcal{A}|$=20

5.1 Implementation of Random MDPs

For each state space size $|\mathcal{S}| \in \{50, 100, 200\}$ and action space size $|\mathcal{A}| \in \{5, 10\}$, we generated 40 instances of randomly generated MDPs. We chose these state and action space sizes to keep computational times reasonable; in preliminary tests, we found that for the same state and action space size, randomly generated MDPs took much longer to solve than the machine maintenance MDPs from §4. For each instance, we randomly generated a stochastic transition matrix P, initial probabilities α, and rewards from the continuous uniform distribution $Unif[0, 1)$. Each row of the transition probability matrix and the vector of initial probabilities were normalized to ensure they summed to 1.

All instances were tested using MMPI (i.e., Algorithm 2) with a maximum number of 50 iterations using the same state ordering rules introduced in

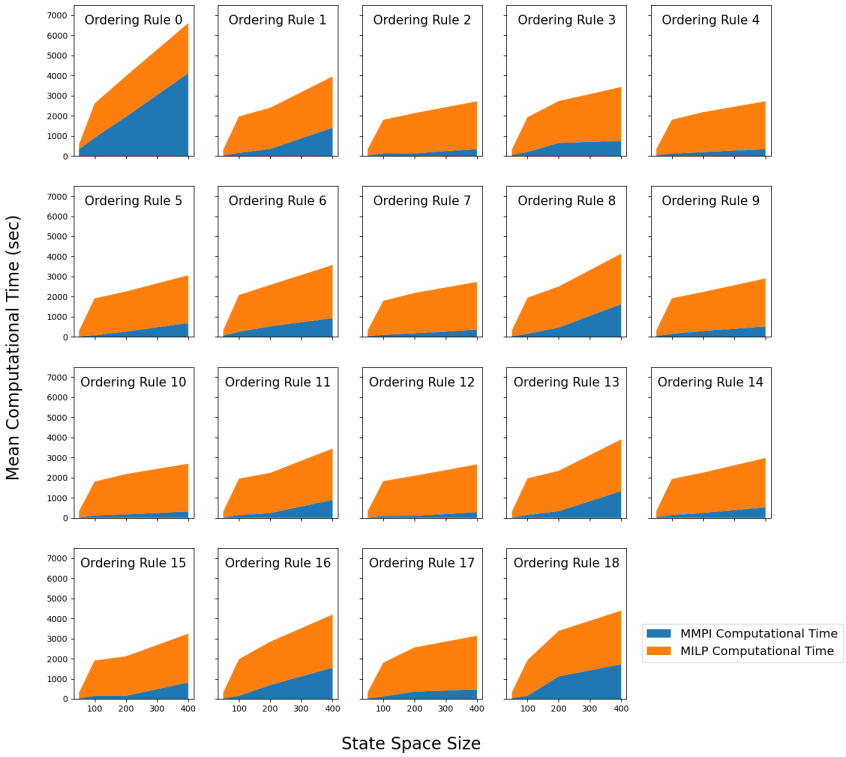

Figure 6 Total computation time for complete solution procedure (MMPI and MILP formulation (6)) for machine maintenance MDPs, $|\mathcal{A}|$=40

§4.2. We evaluated the performance of the MMPI algorithm using the same performance metrics detailed in §4.3.

5.2 Results

Our results on the quality of the policies generated and the computation time taken by the MMPI algorithm using the various state ordering rules are summarized in Figures 7 and 8 for action spaces of size $|\mathcal{A}| = 5$ and $|\mathcal{A}| = 10$, respectively (see Appendix B for details). In each figure, we highlight the state ordering rules on the Pareto frontier of average running time and average optimality gap computed across all instances. As in the machine maintenance setting of §4, there exists a trade-off between the optimality gap of the policies generated and the running time of the state ordering rules. In the random MDP setting, however, this trade-off is more apparent in that the policies produced by the fastest state ordering rules always have among the highest

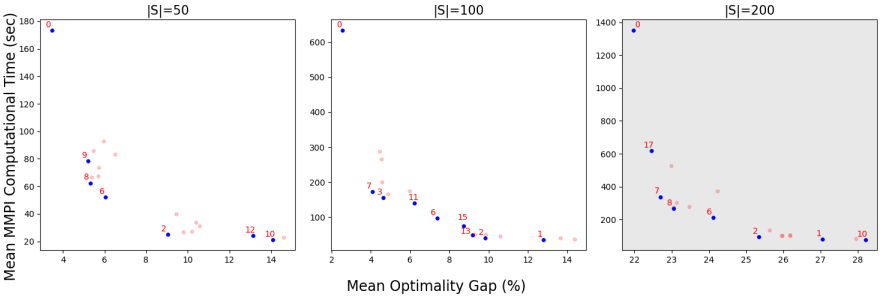

Figure 7 Trade-off curve representation of the mean computational time vs. average optimality gap for various state ordering rules in the MMPI algorithm for random MDPs, $|\mathcal{A}| = 5$. Blue dots (labeled) represent state ordering rules on the Pareto efficiency frontier. Red dots (unlabeled) represent state ordering rules that are dominated. State space(s) for which the monotone optimal policy is unknown for at least one instance (and thus for which the optimality gap was computed using the optimal policy upper bound) are indicated with a gray background.

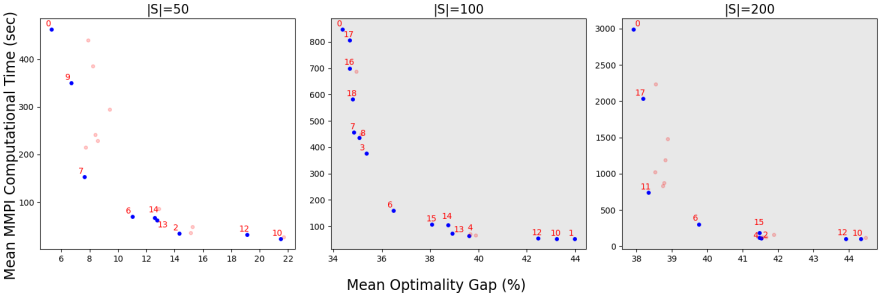

Figure 8 Trade-off curve representation of the mean computational time vs. average optimality gap for various state ordering rules in the MMPI algorithm for random MDPs, $|\mathcal{A}| = 10$. Blue dots (labeled) represent state ordering rules on the Pareto efficiency frontier. Red dots (unlabeled) represent state ordering rules that are dominated. State space(s) for which the monotone optimal policy is unknown for at least one instance (and thus for which the optimality gap was computed using the optimal policy upper bound) are indicated with a gray background.

optimality gaps, as evidenced by the fact that a larger number of state ordering rules fell along the Pareto frontier. Note that, in contrast with the machine maintenance MDPs, it is not guaranteed for random MDPs that there exists an optimal policy which is monotone or almost monotone. Thus, the fastest state ordering rules have higher optimality gaps than in the machine maintenance setting, where even the fastest state ordering rules have relatively low optimality gap; for instances where the solver was not able to find an optimal monotone policy, the optimality gap - calculated using the upper bound value of the optimal policy in these instances - was much higher than in the machine maintenance setting as the optimal policy could be very far from monotone.

Our experiments in the random MDP setting confirm our findings in the machine maintenance setting in that state ordering rule 0 consistently produces policies with the lowest optimality gaps across state ordering rules at the expense of computational time. State ordering rule 0 was able to achieve less

than approximately a 5% optimality gap on average in instances for which the solver was able to find an optimal monotone policy within the time limit. Also in line with our findings from the machine maintenance setting, we found that state ordering rules 6 and 15 were efficient or nearly efficient for all of the state and action space sizes tested; in contrast, however, we found that state ordering rules 3 and 12 were not nearly as consistently efficient. Ordering rules 6, 15, 3, and 12 all consider the quantity $v^*(s) - v_n(s)$, but ordering rules 6 and 15 mediate this quantity with the initial probabilities $\alpha(s)$. This suggests that it provides more benefit to consider $\alpha(s)$ in addition to $v^*(s) - v_n(s)$ in a random MDP setting in which the optimal monotone policy is not guaranteed to be close to monotone. In particular, ordering rule 6 - which prioritizes states with larger initial probabilities - was efficient in all of the state and action space sizes tested. Interestingly, state ordering rules 4 and 13 – the latter of which was proposed in the MPI algorithm by Garcia et al (2022) – were not consistently on the Pareto frontier. Rules 4 and 13 prioritize $\alpha(s)$ in decreasing and increasing order respectively; our results suggest that we can likely improve upon these rules by considering other quantities such as $v^*(s) - v_n(s)$.

We also found that state ordering rules 1 and 10, which respectively order states by decreasing and increasing indices and stay the same from iteration to iteration, are among the fastest state ordering rules across all state and action space sizes. However, comparing state ordering rules with lower computational times and higher optimality gaps, we find diminishing returns along the Pareto frontier when attempting to lower the optimality gap through higher computational times. For example, comparing state ordering rule 10 to rule 6, a decrease in optimality gap of 0.01 corresponds to an increase in computational time of around 5-9 seconds in instances for which the solver was able to find an optimal monotone policy within the time limit; to achieve the same decrease in optimality gap from state ordering rule 6 to rule 0, however, the corresponding increase in computational time is around 40-110 seconds. Thus, in practice, it may be worth using state ordering rule 6 despite the relatively small increase in computational time instead of rule 10 for the improvement in policy quality.

5.3 Computation Time of MILP Formulation with MMPI Warm-start

Figures 9 and 10 depict the total computation time for the solution procedure, i.e., the computation time for the MMPI algorithm to generate a warm start policy then fed into MILP formulation (6), for $|\mathcal{A}| = 5$ and $|\mathcal{A}| = 10$ respectively (see Appendix B for a tabular summary).

We confirm our finding from the machine maintenance setting that the computation time of MILP formulation (6) for each state ordering rule was relatively consistent regardless of the quality of the warm starting solution from MMPI. We found that the solver took much longer to find the optimal policy in the random MDP setting than the machine maintenance setting; while the solver found an optimal monotone policy within the time limit for nearly all instances with $|\mathcal{S}| = 200$, $|\mathcal{A}| = 20$ in the machine maintenance setting, the

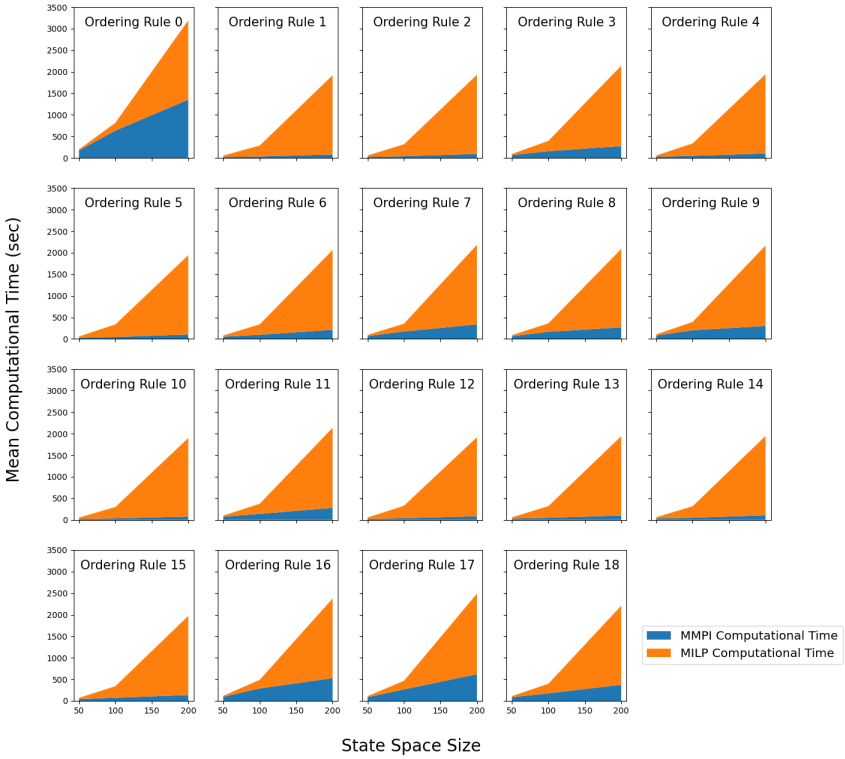

Figure 9 Total computation time for complete solution procedure (MMPI and MILP formulation (6)) for random MDPs, $|\mathcal{A}|$=5

solver was not able to find an optimal monotone policy within the time limit for nearly all instances with $|\mathcal{S}| = 200$, $|\mathcal{A}| = 5$ in the random MDP setting. Furthermore, this finding is consistent given a warm-starting policy of similar quality. In the machine maintenance setting for $|\mathcal{S}| = 100$, $|\mathcal{A}| = 20$, the solver took around 150 seconds on average to solve the MILP when warm-started with policies that had a mean optimality gap of 5%; in the random MDP setting for $|\mathcal{S}| = 100$, $|\mathcal{A}| = 5$, the solver took around 200 seconds on average to solve the MILP when warm-started with policies that had the same mean optimality gap of 5%. We also found that the MMPI algorithm was slower in the random MDP setting than the machine maintenance setting. The fastest state ordering rule was able to solve instances with $|\mathcal{S}| = 200$, $|\mathcal{A}| = 40$ in approximately 100 seconds on average in the machine maintenance setting; in the random MDP setting, the fastest state ordering rule was able to solve instances with $|\mathcal{S}| = 200$ and only $|\mathcal{A}| = 10$ in approximately the same time

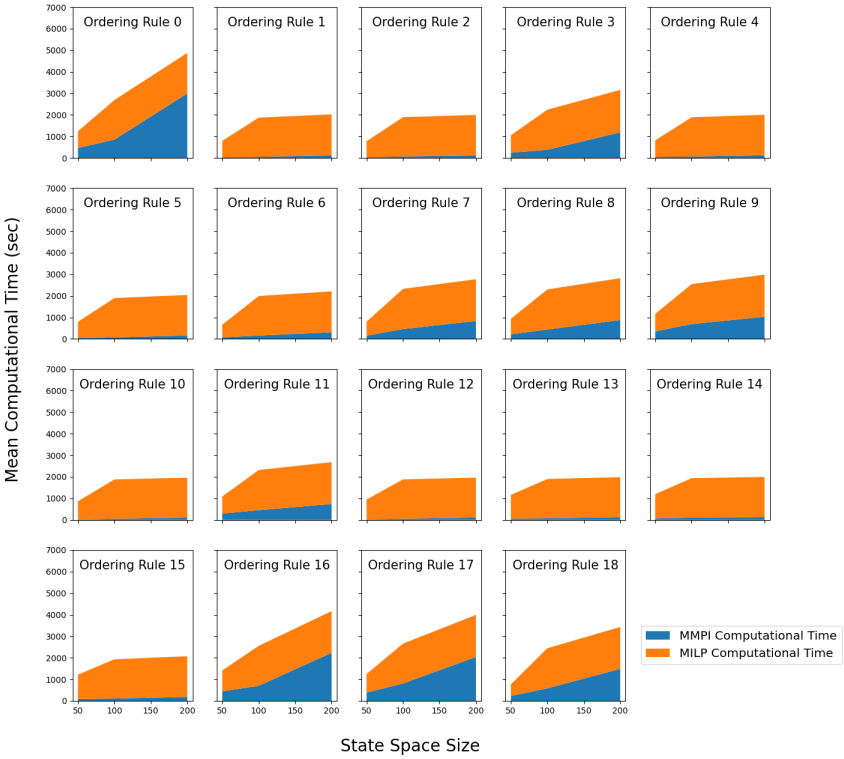

Figure 10 Total computation time for complete solution procedure (MMPI and MILP formulation (6)) for random MDPs, $|\mathcal{A}|$=10

on average. These findings suggest that the structure inherent in the machine maintenance setting is advantageous for both the MMPI algorithm and solving the MILP.

Finally, we confirm our finding that the MMPI algorithm scaled better than the MILP with increasing action space size when warm-started with policies of similar quality for $|\mathcal{S}| = 50$. State ordering rule 10, which was among the fastest state ordering rules for completing the MMPI algorithm, had a negligible increase in computational time for the MMPI algorithm from 21 seconds on average for $|\mathcal{A}| = 5$ to 23 seconds on average for $|\mathcal{A}| = 10$, producing policies with average optimality gaps of 14% and 19% respectively. For the MILP, however, the solver took 25 times longer to solve on average for $|\mathcal{A}| = 10$ than $|\mathcal{A}| = 5$. It is difficult to generalize this finding to larger state spaces, however, as the solver was not able to solve any instances for $|\mathcal{S}| = 100, |\mathcal{A}| = 10$ and $|\mathcal{S}| = 200, |\mathcal{A}| = 10$ within the time limit.

6 Conclusions

In this paper, we proposed an MMPI algorithm with better theoretical convergence and optimality properties than the previously proposed MPI algorithm. We also investigated variations in state ordering rules of the MMPI algorithm and the effects on the quality of the resulting policy as well as computational effort required by the MILP when the resulting policy is used as an initial feasible solution. Computational results reveal that choosing a state ordering rule in the MMPI algorithm involves a trade-off between finding a policy with low optimality gap and finding a policy quickly. This trade-off is more pronounced with random MDPs, as even the fastest state ordering rules in the the machine maintenance setting find high quality policies (less than approximately 5% mean optimality gap). This suggests that in structured problems such as machine maintenance, one can use MMPI as a heuristic algorithm to solve for a near-optimal monotone policy in large-scale MDPs relatively quickly. Our results on random MDPs suggest that both the MMPI algorithm and the MILP formulation benefit computationally from a structured problem, as randomly generated MDPs took much longer to solve than machine maintenance MDPs.

While we show promising results in overcoming computational challenges associated with solving for interpretable policies in sequential decision-making problems, there are some limitations. Our computational results yield interesting insights, but future research may consider deriving theoretical bounds on the quality of the solution found or the rate of convergence of the MMPI algorithm using certain state ordering rules, particularly the ones found to perform well in this paper. Finally, this study focuses entirely on monotone policies, which comprise an important class of interpretable policies. However, we are optimistic that our findings can be adapted to other types of interpretable policies for MDPs such as tree-based policies and more general ordering-based policies. As such, the extensions of our algorithms and analysis to these other types of policies would be a promising direction for future research.

Acknowledgments

The authors would like to thank Lauren Steimle and Wesley Marrero for their input on designing the machine maintenance problem of §3.

Disclosures

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Alagoz O, Maillart LM, Schaefer AJ, et al (2004) The Optimal Timing of Living-Donor Liver Transplantation. Management Science https://doi.org/

10.1287/mnsc.1040.0287

Albright SC (1979) Structural Results for Partially Observable Markov Decision Processes. Operations Research 27(5):1041–1053. https://doi.org/10.1287/opre.27.5.1041

Albright SC, Winston W (1979) Markov Models of Advertising and Pricing Decisions. Operations Research 27(4):668–681. https://doi.org/10.1287/opre.27.4.668

Amram M, Dunn J, Zhuo YD (2022) Optimal policy trees. Machine Learning 111(7):2741–2768. https://doi.org/10.1007/s10994-022-06128-5

Bertsimas D, Klasnja P, Murphy S, et al (2022) Data-driven Interpretable Policy Construction for Personalized Mobile Health. In: 2022 IEEE International Conference on Digital Health (ICDH), pp 13–22, https://doi.org/10.1109/ICDH55609.2022.00010

Boucherie RJ, van Dijk NM (eds) (2017) Markov Decision Processes in Practice, International Series in Operations Research & Management Science, vol 248. Springer International Publishing, Cham, https://doi.org/10.1007/978-3-319-47766-4

Bravo F, Shaposhnik Y (2020) Mining Optimal Policies: A Pattern Recognition Approach to Model Analysis. INFORMS Journal on Optimization 2(3):145–166. https://doi.org/10.1287/ijoo.2019.0026

Chen Q, Ayer T, Chhatwal J (2018) Optimal M-Switch Surveillance Policies for Liver Cancer in a Hepatitis C–Infected Population. Operations Research 66(3):673–696. https://doi.org/10.1287/opre.2017.1706

Ciocan DF, Mišić VV (2022) Interpretable Optimal Stopping. Management Science 68(3):1616–1638. https://doi.org/10.1287/mnsc.2020.3592

de Jonge B, Scarf PA (2020) A review on maintenance optimization. European Journal of Operational Research 285(3):805–824. https://doi.org/10.1016/j.ejor.2019.09.047

Elwany AH, Gebraeel NZ, Maillart LM (2011) Structured Replacement Policies for Components with Complex Degradation Processes and Dedicated Sensors. Operations Research 59(3):684–695. https://doi.org/10.1287/opre.1110.0912

Garcia GGP, Steimle LN, Marrero WJ, et al (2022) Interpretable Policies and the Price of Interpretability in Hypertension Treatment Planning. Optimization Online

Grand-Clément J, Chan C, Goyal V, et al (2021) Interpretable Machine Learning for Resource Allocation with Application to Ventilator Triage. https://doi.org/10.48550/arXiv.2110.10994, 2110.10994

Grinold RC (1973) Technical Note—Elimination of Suboptimal Actions in Markov Decision Problems. Operations Research 21(3):848–851. https://doi.org/10.1287/opre.21.3.848

Hastings NaJ, Mello JMC (1973) Tests for Suboptimal Actions in Discounted Markov Programming. Management Science 19(9):1019–1022. https://doi.org/10.1287/mnsc.19.9.1019

Hu X, Hsueh PYS, Chen CH, et al (2018) An interpretable health behavioral intervention policy for mobile device users. IBM journal of research and development 62(1):4. https://doi.org/10.1147/JRD.2017.2769320

Hu Y, Defourny B (2022) Optimal price-threshold control for battery operation with aging phenomenon: A quasiconvex optimization approach. Annals of Operations Research 317(2):623–650. https://doi.org/10.1007/s10479-017-2505-4

Jardine AK, Lin D, Banjevic D (2006) A review on machinery diagnostics and prognostics implementing condition-based maintenance. Mechanical Systems and Signal Processing 20(7):1483–1510. https://doi.org/10.1016/j.ymssp.2005.09.012

Kaufman DL, Schaefer AJ (2012) Robust Modified Policy Iteration. INFORMS Journal on Computing https://doi.org/10.1287/ijoc.1120.0509

Kim M, Ghate A, Phillips MH (2012) A stochastic control formalism for dynamic biologically conformal radiation therapy. European Journal of Operational Research 219(3):541–556. https://doi.org/10.1016/j.ejor.2011.10.039

Kotas J, Ghate A (2016) Response-guided dosing for rheumatoid arthritis. IIE Transactions on Healthcare Systems Engineering 6(1):1–21. https://doi.org/10.1080/19488300.2015.1126873

Krishnamurthy V (2016) Partially Observable Markov Decision Processes: From Filtering to Controlled Sensing. Cambridge University Press, Cambridge

Kurt M, Kharoufeh JP (2010) Monotone optimal replacement policies for a Markovian deteriorating system in a controllable environment. Operations Research Letters 38(4):273–279. https://doi.org/10.1016/j.orl.2010.03.001

Kurt M, Maillart LM (2009) Structured replacement policies for a Markov-modulated shock model. Operations Research Letters 37(4):280–284. https://doi.org/10.1016/j.orl.2009.03.008

Li B, Si J (2010) Approximate Robust Policy Iteration Using Multilayer Perceptron Neural Networks for Discounted Infinite-Horizon Markov Decision Processes With Uncertain Correlated Transition Matrices. IEEE Transactions on Neural Networks 21(8):1270–1280. https://doi.org/10.1109/TNN.2010.2050334

Liu B, Wu S, Xie M, et al (2017) A condition-based maintenance policy for degrading systems with age- and state-dependent operating cost. European Journal of Operational Research 263(3):879–887. https://doi.org/10.1016/j.ejor.2017.05.006

Lovejoy WS (1987) Some Monotonicity Results for Partially Observed Markov Decision Processes. Operations Research 35(5):736–743. https://doi.org/10.1287/opre.35.5.736

MacQueen J (1967) A Test for Suboptimal Actions in Markovian Decision Problems. Operations Research 15(3):559–561. https://arxiv.org/abs/168468

Mansour Y, Singh S (1999) On the complexity of policy iteration. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, UAI'99, pp 401–408

McKenna RS, Robbins MJ, Lunday BJ, et al (2020) Approximate dynamic programming for the military inventory routing problem. Annals of Operations Research 288(1):391–416. https://doi.org/10.1007/s10479-019-03469-8

McNealey AK, Marrero WJ, Steimle LN, et al (2023) Optimization Methods for Interpretable Treatment and Screening. In: Encyclopedia of Optimization, 3rd edn. Springer, Cham, URL https://doi.org/10.1007/978-3-030-54621-2_866-1

Osaka S (2002) Stochastic Models in Reliability and Maintenance, vol 1. Springer Berlin Heidelberg, Berlin, Heidelberg, https://doi.org/10.1007/978-3-540-24808-8, 1011.1669v3

Perera SC, Sethi SP (2023) A survey of stochastic inventory models with fixed costs: Optimality of (s, s) and (s, s)-type policies—discrete-time case. Production and Operations Management 32(1):131–153

Petrik M, Luss R (2016) Interpretable policies for dynamic product recommendations. In: Conference on Uncertainty in Artificial Intelligence. Association

For Uncertainty in Artificial Intelligence (AUAI)

Powell WB (2016) Perspectives of approximate dynamic programming. Annals of Operations Research 241(1-2):319–356. https://doi.org/10.1007/s10479-012-1077-6

Puterman ML (2014) Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley-Interscience, New York, NY

Puterman ML, Brumelle SL (1979) On the Convergence of Policy Iteration in Stationary Dynamic Programming. Mathematics of Operations Research 4(1):60–69. https://doi.org/10.1287/moor.4.1.60

Rudin C (2019) Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nature Machine Intelligence 1(5):206–215. https://doi.org/10.1038/s42256-019-0048-x, https://arxiv.org/abs/1811.10154

Rudin C, Chen C, Chen Z, et al (2022) Interpretable machine learning: Fundamental principles and 10 grand challenges. Statistics Surveys 16:1–85. https://doi.org/10.1214/21-SS133, https://arxiv.org/abs/2103.11251

Satia JK, Lave RE (1973) Markovian Decision Processes with Uncertain Transition Probabilities. Operations Research 21(3):728–740. https://arxiv.org/abs/169381

Serfozo RF (1976) Monotone optimal policies for Markov decision processes. In: Balinski ML, Beale EML, Dantzig GB, et al (eds) Stochastic Systems: Modeling, Identification and Optimization, II, vol 6. Springer Berlin Heidelberg, Berlin, Heidelberg, p 202–215, https://doi.org/10.1007/BFb0120752

Shechter SM, Bailey MD, Schaefer AJ, et al (2008) The Optimal Time to Initiate HIV Therapy Under Ordered Health States. Operations Research 56(1):20–33. https://doi.org/10.1287/opre.1070.0480

Sinha S, Ghate A (2016) Policy iteration for robust nonstationary Markov decision processes. Optimization Letters 10(8):1613–1628. https://doi.org/10.1007/s11590-016-1040-6

Topin N, Milani S, Fang F, et al (2021) Iterative Bounding MDPs: Learning Interpretable Policies via Non-Interpretable Methods. Proceedings of the AAAI Conference on Artificial Intelligence 35(11):9923–9931. https://doi.org/10.1609/aaai.v35i11.17192

Topkis DM (1978) Minimizing a Submodular Function on a Lattice. Operations Research 26(2):305–321. https://doi.org/10.1287/opre.26.2.305

Ulukus MY, Kharoufeh JP, Maillart LM (2012) Optimal Replacement Policies Under Environment-Driven Degradation. Probability in the Engineering and Informational Sciences 26(03):405–424. https://doi.org/10.1017/S0269964812000083

Wang H (2002) A survey of maintenance policies of deteriorating systems. European Journal of Operational Research 139(3):469–489. https://doi.org/10.1016/S0377-2217(01)00197-7

White DJ (1981) Isotone optimal policies for structured Markov decision processes. European Journal of Operational Research 7(4):396–402. https://doi.org/10.1016/0377-2217(81)90098-9

Yu H, Bertsekas DP (2013) Q-learning and policy iteration algorithms for stochastic shortest path problems. Annals of Operations Research 208(1):95–132. https://doi.org/10.1007/s10479-012-1128-z

Zhang H, Zhang W (2023) Analytical Solution to a Partially Observable Machine Maintenance Problem with Obvious Failures. Management Science 69(7):3993–4015. https://doi.org/10.1287/mnsc.2022.4547

Zheng R, Makis V (2020) Optimal condition-based maintenance with general repair and two dependent failure modes. Computers & Industrial Engineering 141:106,322. https://doi.org/10.1016/j.cie.2020.106322

Appendices

A Machine Maintenance Results

In this appendix, we present detailed computational results for machine maintenance MDPs. Tables 2 and 3 summarize the quality of the policies generated using each state ordering rule in the MMPI algorithm. Tables 4 and 5 and Figures 11 and 12 summarize the computation time for each state ordering rule in the MMPI algorithm while Tables 6 and 7 summarize the number of iterations taken. Tables 8 and 9 summarize computation times for solving the MILP formulation using the policies generated from the MMPI algorithm as a warm start. Table 10 gives an overview of the computational performance for the efficient state ordering rules in the MMPI algorithm. Finally, Table 11 shows the percentage of instances for each state and action space size for which the solver found an optimal solution to the MILP within the time limit.

B Random MDP Results

In this appendix, we present detailed computational results for randomly generated MDPs. Tables 12 and 13 summarize the quality of the policies generated using each state ordering rule in the MMPI algorithm. Tables 14 and 15 and Figures 13 and 14 summarize the computation time for each state ordering

Table 2 Optimality gap (percentage) achieved by each state ordering rule for machine maintenance MDPs with action space size $|\mathcal{A}| = 20$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100$			$\|\mathcal{S}\| = 200$			$\|\mathcal{S}\| = 400$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	**0.1**	**0.0**	0.1	**0.1**	**0.0**	0.2	**0.0**	**0.0**	0.1	**0.0**	**0.0**	0.0
1	11.0	4.2	16.6	4.8	2.3	6.2	3.0	1.0	5.2	2.7	0.8	3.8
2	1.3	1.1	1.2	0.7	0.6	0.6	0.5	0.4	0.4	0.2	0.1	0.2
3	0.5	0.3	0.6	0.3	0.2	0.2	0.1	0.1	0.1	0.1	**0.0**	0.1
4	1.3	1.1	1.2	0.7	0.6	0.6	0.5	0.4	0.4	0.2	0.1	0.2
5	1.3	1.1	1.2	0.7	0.6	0.6	0.5	0.4	0.4	0.2	0.1	0.2
6	0.7	0.4	0.9	0.4	0.2	0.4	0.2	0.2	0.3	0.1	**0.0**	0.1
7	1.3	1.1	1.2	0.7	0.6	0.6	0.5	0.4	0.4	0.1	0.1	0.1
8	5.6	4.0	5.2	3.1	2.2	2.9	1.6	0.9	1.6	0.2	0.2	0.2
9	0.8	0.4	0.8	0.5	0.5	0.3	0.5	0.2	0.7	0.1	**0.0**	0.1
10	1.3	1.1	1.2	0.7	0.6	0.6	0.5	0.4	0.4	0.2	0.1	0.2
11	11.0	4.2	16.6	4.8	2.3	6.2	2.9	1.0	5.2	2.7	0.8	3.8
12	1.2	0.8	1.1	0.7	0.6	0.6	0.4	0.3	0.4	0.1	0.1	0.1
13	11.0	4.2	16.6	4.8	2.3	6.2	3.0	1.0	5.2	2.7	0.8	3.8
14	4.5	3.2	5.4	4.8	2.4	6.2	2.9	1.0	5.2	2.7	0.8	3.8
15	0.9	0.5	0.9	0.6	0.3	0.6	0.2	0.2	0.4	0.3	0.1	0.6
16	7.1	3.8	10.4	3.6	2.2	4.2	1.9	0.9	2.4	0.3	0.1	0.7
17	1.3	1.1	1.2	0.7	0.6	0.6	0.5	0.4	0.4	0.2	0.1	0.2
18	1.6	1.2	1.4	2.9	2.4	2.5	0.4	0.3	0.5	0.3	0.1	0.7

SD=standard deviation

Table 3 Optimality gap (percentage) achieved by each state ordering rule for machine maintenance MDPs with action space size $|\mathcal{A}| = 40$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100^*$			$\|\mathcal{S}\| = 200^*$			$\|\mathcal{S}\| = 400^*$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	**0.1**	**0.1**	0.1	**1.1**	**1.6**	1.2	**0.8**	**1.0**	0.5	**0.4**	**0.4**	0.2
1	36.4	18.9	35.7	21.0	16.0	14.9	12.7	11.6	7.0	5.3	3.8	4.0
2	3.3	2.8	1.8	5.9	6.1	2.1	3.0	3.0	1.0	1.4	1.4	0.4
3	0.8	0.5	0.7	1.7	1.9	1.3	1.1	1.2	0.6	0.6	0.5	0.3
4	3.3	2.8	1.8	5.9	6.1	2.1	3.0	3.0	1.0	1.4	1.4	0.4
5	3.1	2.9	1.8	4.1	4.4	2.2	3.0	3.0	1.0	1.4	1.4	0.4
6	0.9	0.7	0.6	1.9	2.0	1.3	1.2	1.2	0.9	0.6	0.6	0.3
7	3.3	2.8	1.8	5.9	6.1	2.1	3.1	3.1	1.0	1.4	1.3	0.5
8	16.1	15.1	11.0	16.1	14.8	7.7	7.9	7.3	4.5	2.3	1.7	1.5
9	2.2	1.9	1.9	4.3	4.2	2.2	3.3	3.2	1.2	1.3	1.2	0.4
10	3.3	2.8	1.8	5.9	6.1	2.1	3.0	3.0	1.0	1.4	1.4	0.4
11	36.4	18.9	35.7	21.0	16.0	14.9	12.7	11.6	7.0	5.3	3.8	4.0
12	3.0	2.6	1.8	5.2	5.7	2.1	2.6	2.5	1.0	1.2	1.2	0.3
13	36.4	18.9	35.7	21.0	16.0	14.9	12.7	11.6	7.0	5.3	3.8	4.0
14	7.0	3.8	8.1	10.2	9.5	6.6	12.7	10.9	7.0	5.2	3.8	4.1
15	1.3	1.1	0.9	2.9	3.2	1.9	2.0	2.1	0.7	1.3	1.2	0.5
16	16.9	15.8	11.7	17.0	15.5	10.4	7.4	5.4	5.8	3.0	2.5	1.7
17	3.3	2.8	1.8	5.9	6.1	2.1	3.0	3.0	1.0	1.4	1.4	0.4
18	3.7	3.0	2.8	14.2	12.3	8.0	3.0	2.7	1.1	2.3	1.7	1.3

SD=standard deviation

*Not all instances solved to optimality within the time limit (1800 seconds)

rule in the MMPI algorithm while Tables 16 and 17 summarize the number of iterations taken. Table 18 gives an overview of the computational performance for the efficient state ordering rules in the MMPI algorithm. Tables 19 and 20 summarize computation times for solving the MILP formulation using the policies generated from the MMPI algorithm as a warm start. Finally, Table 21 shows the percentage of instances for each state and action space size for which the solver found an optimal solution to the MILP within the time limit.

Table 4 Computation time characteristics (seconds) for each state ordering rule in the MMPI algorithm for machine maintenance MDPs with action space size $|\mathcal{A}| = 20$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100$			$\|\mathcal{S}\| = 200$			$\|\mathcal{S}\| = 400$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	118	106	66	163	159	93	359	302	223	640	591	331
1	39	39	15	96	96	41	220	203	89	509	499	142
2	23	**23**	7	57	61	15	88	81	37	182	200	40
3	50	44	22	107	99	34	174	156	65	331	324	133
4	24	24	8	**56**	59	17	**80**	76	44	181	192	38
5	25	25	6	66	68	15	119	109	40	267	263	37
6	54	50	20	100	84	53	187	177	98	316	301	126
7	24	**23**	9	**56**	**58**	17	**80**	80	34	225	219	52
8	42	41	13	114	104	46	251	232	111	625	570	197
9	40	42	11	96	99	23	189	180	51	310	293	79
10	24	26	7	**56**	60	16	82	73	39	178	191	39
11	35	35	16	88	89	33	213	191	83	423	412	113
12	**22**	**23**	8	58	61	12	84	89	29	180	200	52
13	39	39	15	92	92	37	232	216	94	513	497	143
14	35	33	13	69	73	33	145	124	66	247	250	68
15	34	32	17	83	80	27	119	109	44	383	393	94
16	43	43	13	104	94	42	239	227	93	584	564	201
17	23	**23**	6	**56**	59	16	83	**72**	42	**171**	**172**	33
18	41	37	17	111	112	32	242	235	102	636	591	233

SD=standard deviation

Table 5 Computation time characteristics (seconds) for each state ordering rule in the MMPI algorithm for machine maintenance MDPs with action space size $|\mathcal{A}| = 40$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100$			$\|\mathcal{S}\| = 200$			$\|\mathcal{S}\| = 400$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	333	336	64	905	1060	218	1948	1962	192	4121	3984	481
1	25	25	12	152	150	41	348	321	130	1409	1395	578
2	31	32	11	120	108	37	132	124	41	345	343	87
3	56	54	28	203	181	122	646	589	273	748	716	362
4	31	33	10	125	113	39	193	191	62	342	337	89
5	**21**	**19**	11	**81**	**79**	13	266	267	70	696	696	139
6	64	61	26	260	230	121	529	539	236	933	934	300
7	31	32	10	110	105	39	179	176	55	365	362	112
8	33	31	11	161	151	45	470	434	167	1634	1469	699
9	51	50	14	156	160	35	299	278	93	521	508	102
10	30	33	11	120	103	46	180	187	62	318	319	80
11	26	27	11	144	137	40	242	212	84	904	900	285
12	27	26	9	102	103	32	**109**	**104**	33	**286**	**292**	68
13	26	25	12	154	153	40	335	303	112	1340	1260	574
14	38	39	17	144	137	48	251	266	93	531	559	181
15	27	30	15	153	156	33	154	151	52	820	818	185
16	29	28	13	159	159	41	688	544	439	1556	1415	762
17	31	30	10	121	109	35	365	359	143	469	464	202
18	43	48	17	161	149	57	1121	1039	513	1725	1653	808

SD=standard deviation

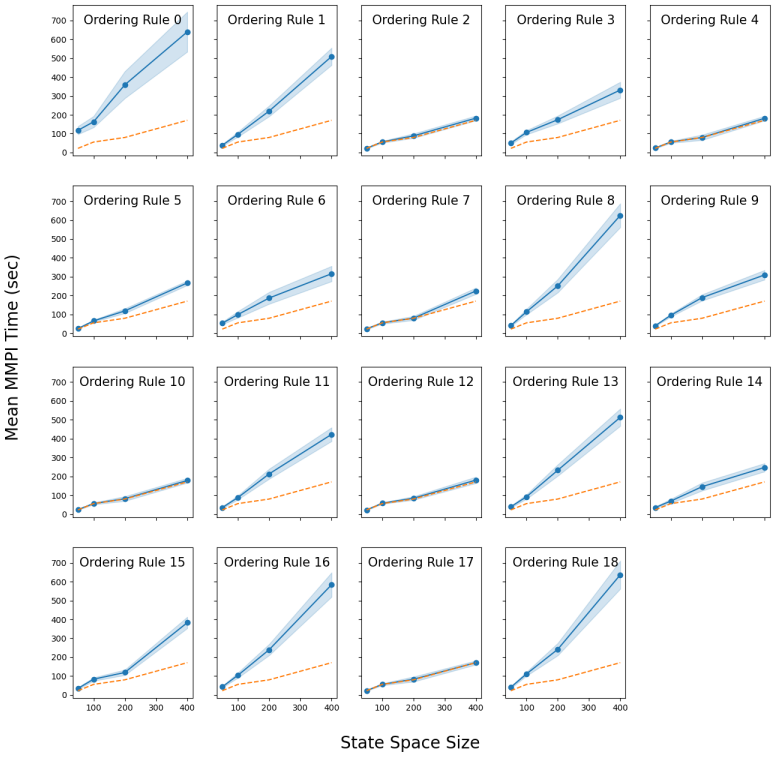

Figure 11 Mean computation time (solid blue line) for MMPI algorithm for machine maintenance MDPs with $|\mathcal{A}| = 20$. Shaded area represents the 95% confidence interval and the dashed orange line represents the minimum mean time taken among all 19 rules at each state space size

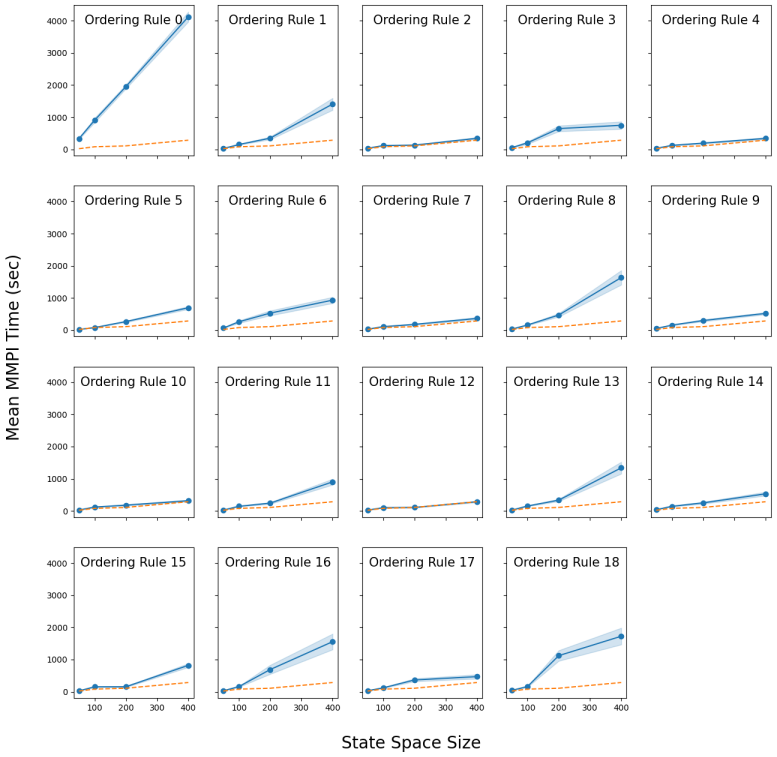

Figure 12 Mean computation time for MMPI algorithm for machine maintenance MDPs with $|\mathcal{A}|$=40. Shaded area represents the 95% confidence interval and the dashed orange line represents the minimum mean time taken among all 19 rules at each state space size

Table 6 Number of iterations for each state ordering rule in the MMPI algorithm for machine maintenance MDPs with $|\mathcal{A}| = 20$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100$			$\|\mathcal{S}\| = 200$			$\|\mathcal{S}\| = 400$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	18.3	15.0	10.7	13.8	12.5	8.0	15.3	15.0	9.7	15.4	14.5	9.9
1	4.8	5.0	1.2	5.2	4.5	2.2	5.1	5.0	2.0	4.8	4.5	1.5
2	3.0	**3.0**	0.6	**3.1**	**3.0**	0.7	3.2	**3.0**	0.8	**3.2**	**3.0**	0.6
3	6.9	6.0	3.0	6.7	6.0	2.9	6.1	5.0	2.4	7.1	6.0	3.2
4	3.0	**3.0**	0.6	**3.1**	**3.0**	0.7	3.2	**3.0**	0.8	**3.2**	**3.0**	0.6
5	**2.9**	**3.0**	0.6	**3.1**	**3.0**	0.7	3.2	**3.0**	0.8	**3.2**	**3.0**	0.6
6	6.9	6.0	3.1	7.3	6.0	3.2	7.1	7.0	2.5	6.5	6.0	2.5
7	3.0	**3.0**	0.6	**3.1**	**3.0**	0.7	3.2	**3.0**	0.8	3.8	4.0	0.9
8	5.0	5.0	1.3	5.4	5.0	2.5	5.3	5.0	2.2	6.0	5.5	1.8
9	5.2	5.0	1.2	5.4	5.0	1.4	5.8	6.0	1.3	5.2	5.0	1.4
10	3.0	**3.0**	0.6	**3.1**	**3.0**	0.7	3.2	**3.0**	0.8	**3.2**	**3.0**	0.6
11	4.8	5.0	1.3	5.2	4.5	2.2	5.1	5.0	2.0	4.8	4.5	1.5
12	**2.9**	**3.0**	0.5	**3.1**	**3.0**	0.5	**3.0**	**3.0**	0.4	3.3	**3.0**	0.6
13	4.8	5.0	1.2	5.2	4.5	2.2	5.1	5.0	2.0	4.8	4.5	1.5
14	5.0	5.0	1.8	5.2	4.5	2.2	5.3	5.0	2.2	4.8	4.5	1.5
15	4.1	4.0	1.7	4.8	5.0	1.0	3.9	4.0	0.9	5.0	5.0	1.2
16	5.0	5.0	1.4	5.4	5.0	2.4	5.2	5.0	2.0	5.6	5.0	1.8
17	3.0	**3.0**	0.6	**3.1**	**3.0**	0.7	3.2	**3.0**	0.8	**3.2**	**3.0**	0.6
18	5.5	5.0	1.8	5.3	5.0	1.8	5.7	5.5	2.1	6.1	6.0	2.0

SD=standard deviation

Table 7 Number of iterations for each state ordering rule in the MMPI algorithm for machine maintenance MDPs with $|\mathcal{A}| = 40$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100$			$\|\mathcal{S}\| = 200$			$\|\mathcal{S}\| = 400$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	43.8	50.5	10.1	49.8	51.0	5.7	50.0	51.0	5.2	50.0	51.0	4.7
1	3.9	3.5	1.5	7.5	7.0	2.7	6.2	6.0	2.0	7.0	6.5	2.5
2	3.3	**3.0**	0.8	3.9	4.0	0.5	4.1	**4.0**	0.9	4.2	**4.0**	0.8
3	7.8	7.0	3.7	14.8	12.0	8.8	15.6	16.0	8.2	13.6	12.0	6.4
4	3.3	**3.0**	0.8	3.9	4.0	0.5	4.1	**4.0**	0.9	4.2	**4.0**	0.8
5	**3.2**	**3.0**	1.3	**3.5**	**4.0**	0.6	4.1	**4.0**	1.0	4.2	4.0	0.8
6	8.3	8.0	3.3	13.9	12.0	6.7	16.9	15.5	7.6	14.7	12.0	7.8
7	3.3	**3.0**	0.7	3.9	4.0	0.5	4.1	**4.0**	0.9	4.3	**4.0**	0.8
8	4.6	4.0	1.5	7.7	7.5	2.8	7.3	7.0	2.3	8.2	8.0	2.9
9	5.7	5.0	1.4	6.0	6.0	1.1	6.9	7.0	1.6	6.0	6.0	1.4
10	3.3	**3.0**	0.8	3.9	4.0	0.5	4.1	**4.0**	0.9	4.2	**4.0**	0.8
11	3.9	3.5	1.5	7.5	7.0	2.5	6.2	6.0	1.9	7.0	6.5	2.6
12	**3.2**	**3.0**	0.8	3.8	**3.5**	1.1	**4.0**	**4.0**	1.1	**4.0**	**4.0**	1.3
13	3.9	3.5	1.5	7.5	7.0	2.7	6.2	6.0	2.0	7.0	6.5	2.5
14	5.3	5.0	1.9	8.1	7.0	3.9	6.3	6.0	1.9	7.5	7.5	2.6
15	4.2	4.0	1.2	6.5	6.0	1.5	4.2	**4.0**	1.2	6.6	6.0	1.5
16	4.6	4.0	1.5	7.7	8.0	2.6	6.8	6.0	2.6	7.8	8.0	3.0
17	3.3	**3.0**	0.7	3.9	4.0	0.5	4.1	**4.0**	0.9	4.2	**4.0**	0.8
18	6.4	6.0	1.7	7.6	7.0	2.6	7.3	7.0	2.3	7.6	7.0	2.7

SD=standard deviation

Table 8 Computation time (seconds) for MILP formulation (6) using warm start policy from MMPI algorithm under each state ordering rule for machine maintenance MDPs with action space size $|\mathcal{A}| = 20$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100$			$\|\mathcal{S}\| = 200$			$\|\mathcal{S}\| = 400$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	115	**70**	278	156	113	279	218	231	69	568	591	187
1	91	86	34	146	103	275	155	161	36	372	**372**	71
2	81	80	25	114	111	28	251	186	376	401	404	57
3	84	86	26	115	112	20	186	185	30	476	479	75
4	82	79	26	114	111	28	252	177	376	402	403	55
5	81	79	25	111	108	29	245	169	379	393	386	53
6	79	79	18	154	108	276	228	189	281	447	442	55
7	76	72	23	115	112	27	256	176	383	407	396	53
8	92	84	35	**98**	**96**	20	**150**	**155**	38	375	392	91
9	86	85	28	125	130	28	180	169	34	430	426	63
10	76	71	24	113	109	26	254	180	378	402	407	55
11	94	91	37	152	110	275	161	168	35	392	382	60
12	80	82	27	111	106	26	213	172	272	412	409	52
13	85	81	37	147	103	276	153	163	38	**369**	376	69
14	95	100	44	154	113	277	186	184	33	420	409	55
15	79	82	24	108	102	32	242	162	382	398	394	62
16	95	91	36	144	101	276	155	159	37	395	418	90
17	**75**	76	23	111	107	26	252	174	380	411	406	59
18	124	81	275	102	100	23	155	162	42	387	406	99

SD=standard deviation

Table 9 Computation time (seconds) for MILP formulation (6) using warm start policy from MMPI algorithm under each state ordering rule for machine maintenance MDPs with action space size $|\mathcal{A}| = 40$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100^*$			$\|\mathcal{S}\| = 200^*$			$\|\mathcal{S}\| = 400^*$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	**235**	**210**	68	1704	1838	434	2022	1999	82	2486	2435	259
1	290	291	63	1812	1859	280	2047	1994	150	2545	2473	228
2	301	287	110	1677	1843	546	1996	1981	55	2370	2374	77
3	262	265	59	1726	1894	485	2084	2064	97	2681	2650	191
4	299	284	109	**1675**	1842	545	1984	1983	42	2378	2386	73
5	284	297	53	1837	**1837**	28	1991	1991	42	**2369**	2358	91
6	270	257	70	1823	1883	291	2058	2077	277	2644	2618	150
7	295	276	107	1684	1845	542	2005	1990	68	**2369**	2366	104
8	296	295	60	1790	1862	301	2040	2033	82	2505	2444	267
9	264	251	66	1762	1857	394	**1937**	1994	380	2386	2382	73
10	300	285	110	1686	1855	541	2000	1998	52	2376	2386	75
11	287	288	63	1809	1859	283	1995	1986	77	2547	2409	293
12	300	297	68	1726	1856	478	1985	1985	76	2373	**2329**	152
13	290	291	62	1813	1857	279	2007	1977	132	2570	2485	279
14	298	300	85	1792	1878	394	2003	1984	74	2448	2427	152
15	283	303	74	1759	1851	412	1961	**1956**	48	2422	2378	225
16	302	303	67	1817	1857	277	2154	2035	301	2641	2552	270
17	297	281	107	1679	1843	546	2194	2037	442	2669	2600	301
18	287	284	67	1775	1869	404	2265	2163	390	2664	2621	362

SD=standard deviation

*Not all instances were solved to optimality within the time limit (1800 seconds)

Table 10 Computational performance of efficient state ordering rules in the MMPI algorithm for machine maintenance MDPs

$\|\mathcal{A}\|$	$\|\mathcal{S}\|$	State Ordering Rule	Optimality Gap (%)		MMPI Time (sec)	
			Mean	SD	Mean	SD
20	50	0	0.1	0.1	118	66
		3	0.5	0.6	50	22
		9	0.8	0.8	40	11
		12	1.2	1.1	22	8
		15	0.9	0.9	34	17
20	100	0	0.0	0.1	359	223
		3	0.1	0.1	174	65
		6	0.5	0.4	80	44
		9	0.4	0.4	84	29
		12	0.0	0.0	0	0
		15	0.0	0.0	0	0
		17	0.0	0.0	0	0
20	200	0	0.0	0.1	359	223
		3	0.1	0.1	174	65
		4	0.5	0.4	80	44
		12	0.4	0.4	84	29
		15	0.2	0.4	119	44
20	400	0	0.0	0.0	640	331
		6	0.1	0.1	316	126
		7	0.1	0.1	225	52
		9	0.1	0.1	310	79
		12	0.1	0.1	180	52
		17	0.2	0.2	171	33
50	50	0	0.1	0.1	333	64
		3	0.8	0.7	56	28
		5	3.1	1.8	21	11
		12	3.0	1.8	27	9
		15	1.3	0.9	27	15
50	100	0	1.1	1.2	905	218
		3	1.7	1.3	203	122
		5	4.1	2.2	81	13
		15	2.9	1.9	153	33
50	200	0	0.8	0.5	1948	192
		3	1.1	0.6	646	273
		6	1.2	0.9	529	236
		12	2.6	1.0	109	33
		15	2.0	0.7	154	52
50	400	0	0.4	0.2	4121	481
		3	0.6	0.3	748	362
		12	1.2	0.3	286	68

SD=standard deviation

Table 11 Percentage of instances for which the solver found an optimal solution to (6) within the time limit for machine maintenance MDPs

State Ordering Rule	$\|\mathcal{A}\| = 20$				$\|\mathcal{A}\| = 40$			
	$\|\mathcal{S}\| = 50$	$\|\mathcal{S}\| = 100$	$\|\mathcal{S}\| = 200$	$\|\mathcal{S}\| = 400$	$\|\mathcal{S}\| = 50$	$\|\mathcal{S}\| = 100$	$\|\mathcal{S}\| = 200$	$\|\mathcal{S}\| = 400$
0	97.5	97.5	100	100	100	22.5	2.5	0
1	100	97.5	100	100	100	5.0	0	5
2	100	100	95.0	100	100	15.0	0	0
3	100	100	100	100	100	22.5	0	0
4	100	100	95.0	100	100	15.0	0	0
5	100	100	95.0	100	100	2.5	0	0
6	100	97.5	97.5	100	100	15.0	2.5	0
7	100	100	95.0	100	100	15.0	0	0
8	100	100	100	100	100	10.0	0	5
9	100	100	100	100	100	10.0	5	0
10	100	100	95.0	100	100	15.0	0	0
11	100	97.5	100	100	100	7.5	0	0
12	100	100	97.5	100	100	15.0	0	0
13	100	97.5	100	100	100	5.0	0	2.5
14	100	97.5	100	100	100	10.0	0	0
15	100	100	95.0	100	100	10.0	0	2.5
16	100	97.5	100	100	100	2.5	0	0
17	100	100	95.0	100	100	15.0	0	0
18	97.5	100	100	100	100	10.0	0	0

Table 12 Optimality gap (percentage) achieved by each state ordering rule for random MDPs with action space size $|\mathcal{A}| = 5$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100$			$\|\mathcal{S}\| = 200^*$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	**5.3**	**4.7**	3.1	**34.4**	**34.2**	2.5	**37.9**	**37.9**	1.4
1	21.7	22.3	5.6	44.0	44.1	2.7	44.5	44.2	2.0
2	14.3	13.9	6.6	39.9	39.7	3.2	41.5	41.0	2.5
3	8.4	7.9	4.2	35.4	35.6	2.4	38.8	38.7	1.4
4	15.3	15.8	6.4	39.6	39.0	3.1	41.5	41.6	2.2
5	15.1	15.3	6.4	39.7	39.1	3.2	41.9	42.0	2.1
6	11.0	11.2	4.3	36.5	36.8	2.9	39.8	39.8	1.7
7	7.6	7.3	3.9	34.8	34.9	2.4	38.7	38.5	1.8
8	7.7	7.4	4.2	35.1	35.9	2.8	38.8	38.8	1.7
9	6.7	6.5	3.9	35.0	35.2	2.7	38.5	38.5	1.5
10	21.5	20.0	7.5	43.2	42.8	2.9	44.3	44.7	3.0
11	9.4	8.3	5.0	35.1	35.2	2.0	38.3	38.1	1.7
12	19.1	17.3	6.9	42.5	42.2	3.4	43.9	44.2	3.0
13	12.7	12.9	5.0	38.9	39.0	3.0	41.7	41.9	1.9
14	12.6	12.9	5.1	38.7	39.0	3.0	41.6	41.7	1.9
15	12.9	13.2	4.9	38.1	38.5	2.8	41.5	41.2	1.7
16	7.9	6.6	5.3	34.7	34.4	2.6	38.5	38.1	1.9
17	8.2	8.0	5.1	34.7	34.8	2.4	38.2	38.0	1.9
18	8.6	8.3	6.9	34.8	34.8	2.3	38.9	38.8	1.9

SD=standard deviation

*Not all instances solved to optimality within the time limit (1800 seconds)

Table 13 Optimality gap (percentage) achieved by each state ordering rule for random MDPs with action space size $|\mathcal{A}| = 10$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100^*$			$\|\mathcal{S}\| = 200^*$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	**5.3**	**4.7**	3.1	**34.4**	**34.2**	2.5	**37.9**	**37.9**	1.4
1	21.7	22.3	5.6	44.0	44.1	2.7	44.5	44.2	2.0
2	14.3	13.9	6.6	39.9	39.7	3.2	41.5	41.0	2.5
3	8.4	7.9	4.2	35.4	35.6	2.4	38.8	38.7	1.4
4	15.3	15.8	6.4	39.6	39.0	3.1	41.5	41.6	2.2
5	15.1	15.3	6.4	39.7	39.1	3.2	41.9	42.0	2.1
6	11.0	11.2	4.3	36.5	36.8	2.9	39.8	39.8	1.7
7	7.6	7.3	3.9	34.8	34.9	2.4	38.7	38.5	1.8
8	7.7	7.4	4.2	35.1	35.9	2.8	38.8	38.8	1.7
9	6.7	6.5	3.9	35.0	35.2	2.7	38.5	38.5	1.5
10	21.5	20.0	7.5	43.2	42.8	2.9	44.3	44.7	3.0
11	9.4	8.3	5.0	35.1	35.2	2.0	38.3	38.1	1.7
12	19.1	17.3	6.9	42.5	42.2	3.4	43.9	44.2	3.0
13	12.7	12.9	5.0	38.9	39.0	3.0	41.7	41.9	1.9
14	12.6	12.9	5.1	38.7	39.0	3.0	41.6	41.7	1.9
15	12.9	13.2	4.9	38.1	38.5	2.8	41.5	41.2	1.7
16	7.9	6.6	5.3	34.7	34.4	2.6	38.5	38.1	1.9
17	8.2	8.0	5.1	34.7	34.8	2.4	38.2	38.0	1.9
18	8.6	8.3	6.9	34.8	34.8	2.3	38.9	38.8	1.9

SD=standard deviation

*Not all instances solved to optimality within the time limit (1800 seconds)

Table 14 Computation time characteristics (seconds) for each state ordering rule in the MMPI algorithm for random MDPs with $|\mathcal{A}| = 5$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100$			$\|\mathcal{S}\| = 200$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	174	173	82	634	604	285	1352	1483	446
1	23	21	5	**35**	38	15	81	78	13
2	25	26	6	41	**37**	12	94	97	19
3	68	62	27	155	141	66	275	279	111
4	27	27	10	51	54	19	103	107	19
5	27	27	10	46	45	19	102	108	22
6	52	48	16	97	100	30	212	220	47
7	66	58	34	172	166	87	338	332	132
8	62	60	28	166	136	90	268	261	105
9	78	74	34	201	174	101	302	289	139
10	**21**	**20**	4	36	38	17	**77**	**77**	15
11	74	70	28	139	128	65	279	253	104
12	24	21	9	40	39	17	84	78	15
13	31	28	10	49	44	20	101	107	21
14	34	30	13	50	52	22	106	110	21
15	40	40	10	75	72	23	136	144	20
16	93	82	55	288	239	164	528	446	320
17	86	81	41	266	226	139	619	575	293
18	83	64	53	175	172	96	373	286	260

SD=standard deviation

Table 15 Computation time characteristics (seconds) for each state ordering rule in the MMPI algorithm for random MDPs with $|\mathcal{A}| = 10$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100$			$\|\mathcal{S}\| = 200$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	463	496	147	848	929	213	2994	2826	279
1	26	24	9	**53**	55	9	117	117	34
2	34	35	11	67	64	23	114	113	39
3	241	224	118	376	341	236	1187	1224	542
4	48	38	26	64	66	17	120	120	31
5	35	34	16	72	68	29	160	160	54
6	70	62	31	159	149	53	306	289	131
7	153	114	108	457	389	296	834	731	379
8	215	192	172	436	349	317	876	822	476
9	351	326	223	689	637	340	1027	1090	441
10	**23**	**22**	12	**53**	**51**	12	**104**	**105**	23
11	295	198	231	451	363	246	739	686	370
12	32	26	21	54	53	21	107	112	25
13	62	48	43	73	74	20	127	122	29
14	67	59	33	105	100	31	129	121	34
15	86	75	42	108	114	21	188	186	50
16	440	366	301	700	850	295	2232	2491	866
17	386	372	214	807	927	313	2038	2160	736
18	229	200	130	583	584	310	1484	1734	634

SD=standard deviation

Table 16 Number of iterations for each state ordering rule in the MMPI algorithm for random MDPs with $|\mathcal{A}| = 5$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100$			$\|\mathcal{S}\| = 200$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	20.2	18.5	11.1	34.4	37.0	14.6	40.0	51.0	15.0
1	2.5	**2.0**	0.6	**2.2**	**2.0**	0.4	2.2	**2.0**	0.4
2	2.7	3.0	0.7	2.4	**2.0**	0.5	2.6	3.0	0.6
3	7.7	7.0	3.1	9.4	9.0	3.9	9.3	9.0	3.4
4	3.0	3.0	1.1	3.2	3.0	1.0	2.9	3.0	0.8
5	3.0	3.0	1.0	3.0	3.0	1.0	2.9	3.0	0.9
6	5.7	5.0	2.1	5.9	6.0	1.7	6.4	6.0	1.7
7	7.2	7.0	3.6	10.0	9.0	5.4	10.9	10.0	5.2
8	6.8	6.0	3.3	9.7	7.5	6.2	8.6	8.0	4.2
9	8.9	8.0	4.1	11.6	9.0	6.5	11.4	10.0	5.6
10	**2.4**	**2.0**	0.5	2.3	**2.0**	0.5	**2.1**	**2.0**	0.4
11	8.1	8.0	3.2	8.7	8.0	4.4	9.2	8.0	4.2
12	2.6	**2.0**	1.0	2.7	3.0	0.8	2.3	**2.0**	0.5
13	3.4	3.0	1.1	3.0	3.0	1.1	2.9	3.0	0.8
14	3.6	3.0	1.3	3.1	3.0	1.1	3.0	3.0	0.9
15	4.3	4.0	1.1	4.0	4.0	0.9	3.7	4.0	0.8
16	10.2	9.0	6.1	15.7	13.0	9.1	18.2	17.0	10.9
17	9.6	9.0	4.6	15.6	13.5	8.2	19.9	17.5	10.6
18	9.5	7.0	6.3	11.3	9.5	6.1	12.7	10.0	8.7

SD=standard deviation

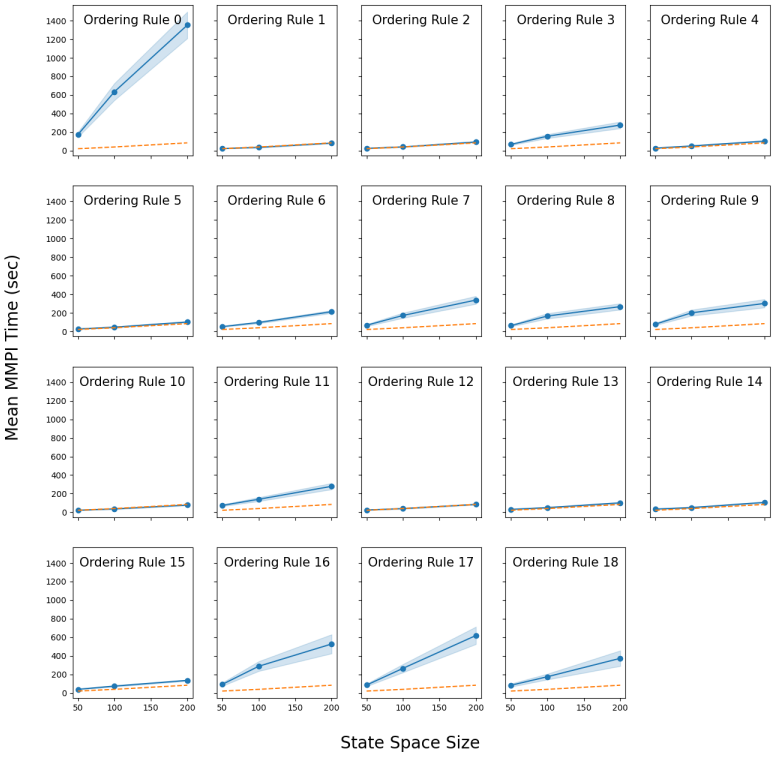

Figure 13 Mean computation time for MMPI algorithm for random MDPs with $|\mathcal{A}|$=5. Shaded area represents the 95% confidence interval and the dashed orange line represents the minimum mean time taken among all 19 rules at each state space size

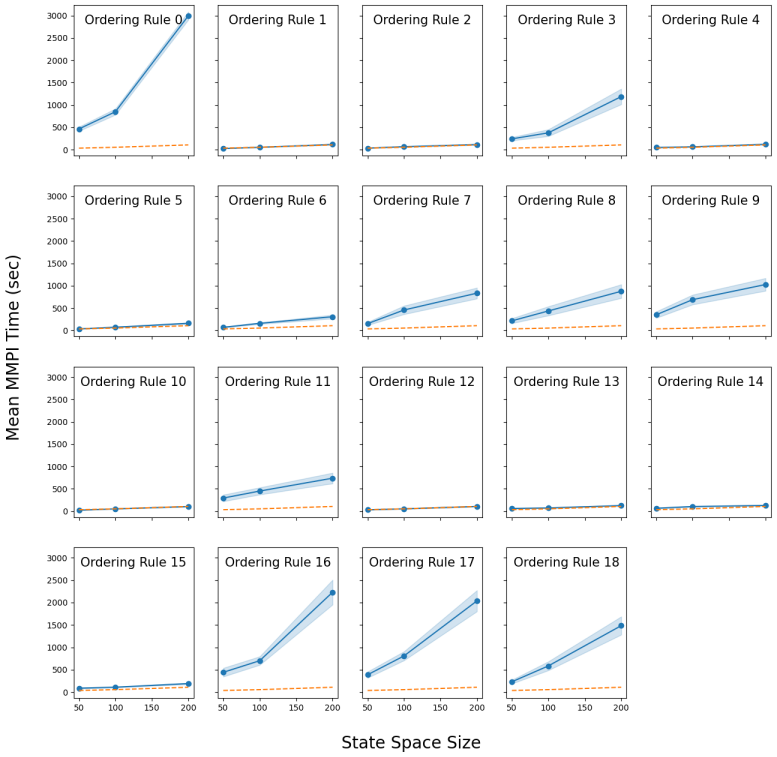

Figure 14 Mean computation time for MMPI algorithm for random MDPs with $|\mathcal{A}|$=10. Shaded area represents the 95% confidence interval and the dashed orange line represents the minimum mean time taken among all 19 rules at each state space size

Table 17 Number of iterations for each state ordering rule in the MMPI algorithm for random MDPs with $|\mathcal{A}| = 10$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100$			$\|\mathcal{S}\| = 200$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	42.6	51.0	12.8	45.5	51.0	12.1	50.6	51.0	2.5
1	**2.5**	**2.0**	0.6	2.6	3.0	0.5	2.5	**2.0**	0.5
2	3.5	3.0	0.9	3.6	3.0	1.6	3.2	3.0	0.9
3	17.2	16.0	7.9	20.9	18.0	13.0	25.1	23.5	12.9
4	3.8	4.0	1.3	3.4	3.0	1.1	3.1	3.0	0.9
5	4.0	4.0	1.4	3.9	3.0	2.0	3.4	3.0	1.1
6	7.9	7.0	3.2	8.6	8.0	3.3	9.9	9.0	4.8
7	15.8	13.0	11.2	24.7	22.0	15.7	27.1	24.0	14.7
8	16.5	13.0	12.2	22.9	18.0	15.9	28.7	30.0	15.6
9	19.5	18.0	11.3	30.1	28.0	15.8	31.6	32.0	15.3
10	2.6	3.0	0.6	**2.4**	**2.0**	0.5	**2.4**	**2.0**	0.5
11	18.2	14.5	11.4	23.2	22.0	12.1	26.1	21.5	13.3
12	3.0	3.0	1.2	2.9	3.0	0.9	3.0	3.0	1.0
13	4.0	3.0	1.9	3.6	3.0	1.1	3.4	3.0	1.1
14	4.1	3.0	1.9	3.9	4.0	1.5	3.8	3.0	1.5
15	4.9	4.5	1.5	4.9	5.0	1.3	4.4	4.0	1.1
16	24.0	21.0	15.3	35.6	44.5	16.5	41.2	51.0	15.9
17	21.6	21.0	11.6	39.0	51.0	16.0	42.1	51.0	15.3
18	25.5	22.0	13.7	31.1	30.5	16.0	36.2	43.5	16.4

SD=standard deviation

Table 18 Computational performance of efficient state ordering rules in the MMPI algorithm for random MDPs

$\|\mathcal{A}\|$	$\|\mathcal{S}\|$	State Ordering Rule	Optimality Gap (%)		MMPI Time (sec)	
			Mean	SD	Mean	SD
5	50	0	3.5	3.2	174	82
		2	9.0	5.8	25	6
		6	6.0	4.2	52	16
		8	5.3	4.0	62	28
		9	5.2	4.1	78	34
		10	14.1	7.3	21	4
		12	13.1	7.3	24	9
5	100	0	2.6	1.6	634	285
		1	12.8	5.5	35	15
		2	9.8	5.4	41	12
		3	4.6	2.7	155	66
		6	7.4	3.3	97	30
		7	4.1	3.8	172	87
		11	6.2	3.6	139	65
		13	9.2	3.9	49	20
		15	8.7	3.6	75	23
5	200	0	22.0	1.6	1352	446
		1	27.1	15.2	81	13
		2	25.3	15.2	94	19
		6	24.1	15.2	212	47
		7	22.7	16.0	338	132
		8	23.1	15.7	268	105
		10	28.2	14.6	77	15
		17	22.5	16.0	619	293
10	50	0	5.3	3.1	463	147
		2	14.3	6.6	34	11
		6	11.0	4.3	70	31
		7	7.6	3.9	153	108
		9	6.7	3.9	351	223
		10	21.5	7.5	23	12
		12	19.1	6.9	32	21
		13	12.7	5.0	62	43
		14	12.6	5.1	67	33
10	100	0	34.4	2.5	848	213
		1	44.0	2.7	53	9
		3	35.4	2.4	376	236
		4	39.6	3.1	64	17
		6	36.5	2.9	159	53
		7	34.8	2.4	457	296
		8	35.1	2.8	436	317
		10	43.2	2.9	53	12
		12	42.5	3.4	54	21
		13	38.9	3.0	73	20
		14	38.7	3.0	105	31
		15	38.1	2.8	108	21
		16	34.7	2.6	700	295
		17	34.7	2.4	807	313
		18	34.8	2.3	583	310
10	200	0	37.9	1.4	2994	294
		2	41.5	2.5	114	39
		4	41.5	2.2	120	31
		6	39.8	1.7	306	131
		10	44.3	3.0	104	23
		11	38.3	1.7	739	370
		12	43.9	3.0	107	25
		15	41.5	1.7	188	50
		17	38.2	1.9	2038	736

SD=standard deviation

Table 19 Computation time (seconds) for MILP formulation (6) using warm start policy from MMPI algorithm under each state ordering rule for random MDPs with action space size $|\mathcal{A}| = 5$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100$			$\|\mathcal{S}\| = 200^*$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	**25**	29	10	**182**	**165**	70	1841	1876	134
1	33	33	10	252	263	72	1838	1836	15
2	31	32	9	276	299	72	1836	1835	12
3	**25**	**27**	8	239	247	61	1865	1855	27
4	30	31	8	285	298	70	1840	1837	14
5	30	30	8	290	297	70	1842	1843	12
6	28	28	9	241	247	68	1850	1849	14
7	**25**	**27**	8	183	170	74	1845	1860	92
8	26	28	7	194	183	63	**1820**	1856	191
9	26	28	8	194	187	60	1867	1861	22
10	32	33	8	263	274	71	1824	**1830**	43
11	**25**	**27**	8	236	223	75	1858	1857	31
12	31	31	8	290	304	74	1835	1835	15
13	29	29	6	269	270	67	1843	1841	14
14	29	28	5	266	267	65	1843	1843	14
15	30	29	5	267	265	65	1838	1837	28
16	**25**	**27**	8	201	205	73	1848	1872	161
17	**25**	**27**	9	201	189	66	1869	1866	48
18	27	28	11	221	216	88	1835	1877	171

SD=standard deviation

*Not all instances solved to optimality within the time limit (1800 seconds)

Table 20 Computation time (seconds) for MILP formulation (6) using warm start policy from MMPI algorithm under each state ordering rule for random MDPs with action space size $|\mathcal{A}| = 10$

State Ordering Rule	$\|\mathcal{S}\| = 50$			$\|\mathcal{S}\| = 100^*$			$\|\mathcal{S}\| = 200^*$		
	Mean	Median	SD	Mean	Median	SD	Mean	Median	SD
0	777	522	554	1840	1837	12	1883	1869	46
1	766	535	463	**1818**	**1817**	7	1905	1902	42
2	743	450	484	1826	1823	12	1881	1869	36
3	811	663	506	1862	1862	19	1971	1961	71
4	769	641	527	1824	1823	9	1883	1875	32
5	760	579	529	1826	1825	11	1881	1869	39
6	590	384	394	1832	1831	10	1901	1889	33
7	652	365	503	1865	1861	21	1939	1933	55
8	717	530	539	1862	1858	17	1942	1940	45
9	811	545	536	1858	1860	22	1956	1946	50
10	842	565	551	1825	1822	11	**1854**	**850**	24
11	796	573	521	1864	1863	23	1944	1933	43
12	915	688	575	1826	1826	10	1855	1850	22
13	1100	776	548	1825	1825	9	1857	1851	23
14	1127	884	546	1833	1833	16	1866	1860	23
15	1134	857	536	1823	1821	9	1884	1876	30
16	972	696	543	1855	1853	18	1935	1935	66
17	862	654	535	1852	1839	37	1953	1945	63
18	**540**	**297**	446	1860	1858	18	1946	1936	63

SD=standard deviation

*Not all instances solved to optimality within the time limit (1800 seconds)

Table 21 Percentage of instances for which the solver found an optimal solution to (6) within the time limit for random MDPs

State Ordering Rule	$\|\mathcal{A}\| = 5$			$\|\mathcal{A}\| = 10$		
	$\|\mathcal{S}\| = 50$	$\|\mathcal{S}\| = 100$	$\|\mathcal{S}\| = 200$	$\|\mathcal{S}\| = 50$	$\|\mathcal{S}\| = 100$	$\|\mathcal{S}\| = 200$
0	100	100	17.5	87.5	0	0
1	100	100	0	97.5	0	0
2	100	100	0	95.0	0	0
3	100	100	0	90.0	0	0
4	100	100	0	95.0	0	0
5	100	100	0	97.5	0	0
6	100	100	0	100	0	0
7	100	100	7.5	100	0	0
8	100	100	7.5	90.0	0	0
9	100	100	0	90.0	0	0
10	100	100	5.0	92.5	0	0
11	100	100	2.5	90.0	0	0
12	100	100	2.5	87.5	0	0
13	100	100	0	75.0	0	0
14	100	100	0	72.5	0	0
15	100	100	2.5	72.5	0	0
16	100	100	10.0	87.5	0	0
17	100	100	7.5	92.5	0	0
18	100	100	10.0	100	0	0