Markov Decision Processes with Sequential Sensor Measurements

Mahmoud El Chamie\(^{1}\), Dylan Janak\(^{2}\), and Behcet Acıkmese\(^{2}\)

Abstract

Markov Decision Processes (MDPs) have been used to formulate many decision-making problems in science and engineering. The objective is to synthesize the best decision (action selection) policies to maximize expected rewards (or minimize costs) for a stochastic dynamical system. In this paper, we introduce a new type of sensor measurement to the MDP model that provides additional information about the stochastic process, and hence that information can be incorporated in the decision policy to increase the performance. In this model, the additional measurement estimates the possible state transition if a particular action in a prescribed sequential order is taken, i.e., we have more refined information on the possible transition in real-time before taking an action. This new MDP model with the additional sequential measurements is referred to as sequentially-observed MDP (SO-MDP). We show that the SO-MDP shares some similar properties with a standard MDP; among randomized history dependent policies, deterministic Markovian policies are still optimal. Optimal SO-MDP policies have the advantage of producing better rewards than standard MDP policies due to the additional measurements, however computing these policies is more complex. We present two algorithms for solving the finite-horizon SO-MDP problem: the first algorithm is based on linear-programming, and the second algorithm is based on dynamic programming. We show that the complexity of computing optimal policies of the SO-MDP model with perfect sensors is the same as standard MDP. Simulations demonstrate that the SO-MDP model outperforms the standard MDP model in the presence of high environment uncertainty.

Key words: Markov Decision Process (MDP); Decision Theory; Finite Horizon Sequentially-Observed MDP (SO-MDP); Linear Programming Based Synthesis; Sequential Sensor Measurements

1 Introduction

Markov Decision Processes (MDPs) have been used to formulate many decision-making problems in a variety of areas of science and engineering [2–4]. MDPs have proved useful in modeling decision-making problems for stochastic dynamical systems where the dynamics cannot be fully captured using first principles formulations.

MDP models can be constructed by utilizing the available data, which allows determination of state transition probabilities. Hence MDPs play a critical role in big-data analytics. Indeed very popular methods of machine learning such as reinforcement and its variants [5] [6] are built on the MDP framework. With the increased interest and efforts in Cyber-Physical Systems (CPS), there is even more interest in MDPs to facilitate rigorous construction of innovative hierarchical decision-making architectures, where the MDP framework can integrate physics-based and data-driven models. Such decision architectures can utilize a systematic approach to bring physical devices together with software to benefit many emerging engineering applications, such as autonomous systems.

This paper aims to extend the standard MDP framework by designing policies that leverage a new type of additional sensed information. In particular, we consider a scenario where not only the current state of the agent is known (full-state feedback) but also the transition due to an action can be measured by sensors in a sequential manner: the outcome of the first available action is observed and a decision is made on whether to take the action or not, and this process is continued until one of

---

\(^{*}\) The material in this paper was partially presented at the 2016 American Control Conference (ACC 2016) [1]. This paper is substantially different from the conference publication [1] with new theoretical contributions (Lemma 1, Proposition 1), new technical contributions (the dynamic programming algorithm, Algorithm 3, and its complexity analysis), and higher fidelity simulations with large state space implementation.

Email addresses: elchamm@utrc.utc.com (Mahmoud El Chamie), djanak@uw.edu (Dylan Janak), behcet@uw.edu (Behcet Acıkmese).

1 M. El Chamie is with the Systems Department, United Technologies Research Center, 411 Silver Ln, East Hartford, CT 06108, USA.
2 D. Janak and B. Acıkmese are with the University of Washington, Department of Aeronautics and Astronautics, Seattle, WA 98195, USA.
the actions (in a prescribed order) is taken. Decisions are taken at instances called phases. A phase starts with an observation for the transition caused by an action and ends with a decision about whether to take this action or not.

We refer to this model with the extended set of sensed information as a sequentially-observed MDP (SO-MDP) model. We show in this paper that among history dependent policies, Markovian policies are still optimal for SO-MDPs, whose proof utilizes similar arguments as those of the standard MDP. We also prove that there exists an optimal deterministic policy for which we provide a synthesis method. The main advantage of SO-MDP policies is that they produce better rewards than standard optimal MDP policies by benefiting from the additional observations. The policy synthesis for SO-MDPs, on the other hand, is more complex because the synthesis problem is not convex in the decision variables. We provide two algorithms that solve the SO-MDP problem. The first algorithm shows that, via an appropriate change of variables, a linear-programming-based method can be formulated to synthesize optimal SO-MDP decision-making policies. The second algorithm solves the MDP by extending the state-space and applying a dynamic programming algorithm whose complexity is of the same order as the standard MDP in the presence of perfect sensors. The introduced model and the synthesis results are also demonstrated on an agent exploration example.

2 Related Work

MDPs have been widely studied since the pioneering work of Bellman [7], which provided the foundation of dynamic programming, and the book of Howard [8] that popularized the study of decision processes. The standard MDP models are applied to diverse fields including robotics, automatic control, economics, manufacturing, and communication networks. There have been several extensions and generalizations of the MDP models to fit specific application requirements and considerations into the models. Standard MDP problems consider a full-state feedback model where at every decision epoch, agents know their current state and the reward for choosing an action, while the environment is stochastic, i.e., the transitions cannot be predicted in a deterministic manner. On the other hand, partially observed MDPs (POMDPs) relax the full-state feedback assumption to take into account uncertainties in the agent state knowledge [9]. The SO-MDP proposed in this paper is fundamentally different from the POMDP model because we assume full-state feedback (as in the standard MDP) and the next-state is the one that is partially observed through sensor measurements. This is an important difference because with our formulation, we can solve for optimal policies in polynomial time algorithms, as compared to POMDPs that are NP-hard problems.

Other models consider uncertainties in state transition/reward functions. Learning methods are developed to handle such uncertainties (e.g., reinforcement learning [10]). When learning methods are effective, closed-loop control is usually used to provide better performance. In the absence of the stochastic nature (deterministic systems), open-loop and closed-loop performance are equivalent [11]. In standard MDPs, decisions are taken on discrete epochs. Continuous-time MDPs [12] extend this model by relaxing the assumption of discrete epochs into a continuum of times in which events can occur. Another extension is the Bandit problem [13], where the agents can observe the random reward of different actions and have to choose the actions that maximize the sum of rewards through a sequence of repeated experiments. In other decision-making problems, determination of optimal stopping time is studied to determine the optimal epoch for a particular action [14, Chapter 13]. In other applications, multi-objective cost functions or constraints are considered for the computation of the optimal MDP policies [15].

In most of the relevant literature, the extensions to the standard MDP models are obtained by relaxing some of its assumptions (like observability of current state, known rewards, transition probabilities, etc.). In this paper, however, we extend standard MDP problems by considering a more general model when more information about the environment and the process is available. This latter assumption is motivated by the fact that the evolving field of Internet of Things (IoT) is providing agents with a lot of additional data that can be utilized in the model to synthesize better decision-making strategies. In particular, we assume that not only the current state, but also the environmental transition due to possible actions are also observed in a sequential manner. We aim to build decision-making models that benefit from this class of information to generate policies which have better total expected rewards. The additional information that can be fruitfully exploited for obtaining more effective decision policies.

In our recent paper [16] we proposed a Markov decision model with only two actions (ON and OFF) which also has observations of action outcomes. However there are two fundamental differences. First, our goal in [16] was to synthesize the underlying Markov chain to converge to a desired state probability distribution within safety constraints, whereas our goal here is to solve an MDP with a cost/reward metric rather than controlling the state probability distribution of the resulting Markov chain. Second, in [16], the outcome of only a single action could be observed at any time epoch, whereas here we can observe through (possibly imperfect) sensors all the outcomes within a prescribed order. The work in this paper is also related to the literature on Markovian jump models and control [17, 18].
3 Sequentially Observed MDP

In this section we compare decision policies of the standard MDP to those of the introduced SO-MDP model.

3.1 Stochastic System

The dynamic behavior of a one-dimensional stochastic system [11, p. 10] is modeled by an equation of the form:

\[ X_{t+1} = f_t(X_t, A_t, \omega_t), \quad t = 1, \ldots, N - 1 \quad (1) \]

where \( t \) is the decision epoch (or time instant) of horizon \( N \), \( X_t \) is the state, and \( A_t \) is the action chosen by the decision maker. We assume in this paper that the states and actions can have a finite number of possibilities, i.e., finite MDP. The action \( A_t \) is selected according to a control law (decision rule) to be designed. \( \omega_t \) is a random variable that characterizes the stochastic nature of the process. In MDP problems, every sequence of actions \( \{A_t; t \geq 1\} \) induces a discrete time Markov process \( \{X_t; t \geq 1\} \). Thus MDPs are also called controlled Markov chains.

3.2 Policy Description

A randomized decision rule defines a probability distribution over the action set \( \mathbb{P}[A] \) where \( A = \{a_1, \ldots, a_m\} \) is the set of actions. A Markovian policy for standard MDP is a sequence of predefined decision rules from \( t = 1 \) to \( t = N - 1 \) where each rule at time \( t \) maps \( X_t \) to a probability distribution over the set of actions without using the state-action information from the history \( t - 1, \ldots, 1 \). In particular, under the assumption of a full state feedback, at any decision epoch \( t \) the system observes the state \( X_t \) and applies an action \( A_t \) given by the decision rule. Figure 1 provides the standard controlled Markov chain approach for the standard MDP system. Note that, it can be shown that optimal policies for standard MDPs are Markovian and deterministic [19].

In SO-MDP, we assume that actions are ordered in a prescribed sequence. Given the current state \( X_t \), the system goes over the ordered sequence of actions. When the system reaches an action \( a_k \), the next transition state \( X_{t+1} \) can be measured by a sensor observation \( (O_t, Z_t) \) before taking the action. Then the decision rule is a probability distribution \( \mathbb{P}[(\text{Yes}, \text{No})] \) over acceptance/rejection of the observed transition. The yes/no decisions are taken at instances called phases. A phase \( Z_t \) starts with a sensor observation for the transition outcome of an action at time \( t \) and ends with a decision about whether to take the action, or reject it to make another observation. A Markovian policy for SO-MDP is a predefined sequence of decision rules that map \( \{X_t, O_t, Z_t\} \) to a probability distribution over each of the yes/no, acceptance/rejection, possibilities (see Figure 2). In SO-MDP, the order of the actions \( a_1, \ldots, a_m \) can affect the optimal policy because they are observed and acted upon sequentially. For \( N \) decision epochs, there can be up to \( (m!)^N \) possible orderings of actions where \( ! \) is the factorial operation. The results in this paper apply to any particular fixed ordering, and we then provide a heuristic that relies on standard MDP Q-function to compute an ordering that maximizes a lower bound on the optimal performance metric of SO-MDP.

Remark 1 Any given MDP model can be extended to SO-MDP model by adding extra sensing capabilities to the system, and hence improve the overall performance of the constructed policies.

3.3 Mathematical Formulation of SO-MDPs

3.3.1 States and Actions

Let the set \( S = \{1, \ldots, n\} \) be the set of states having a cardinality \(|S| = n\). Let us define \( A_s = \{a_1, \ldots, a_m\} \) to be the set of actions available in state \( s \). Without loss of generality the number of actions does not change with the state, i.e., \(|A_s| = m\) for any \( s \in S \), hence the set of actions is simply denoted by \( A \). We consider a discrete-time system where actions are taken at different decision epochs. Let \( X_t, A_t \), and \( Z_t \) be respectively the random variables corresponding to state, action, and the phase at the \( t \)-th decision epoch.
3.3.2 Decision Rule and Policy

The history of the process is given by:

\[ h_t = \{s_1, y_1, s_2, y_2, \ldots, s_{t-1}, y_{t-1}, s_t \} \]

where \( s_k \) and \( y_k \) are respectively realizations of the random variables \( X_k \) and \( A_k \). Let \( H_t \) be the set for all possible realizations, i.e., \( h_t \in H_t \). We define a randomized decision rule \( d_t \) at time \( t \) for the standard MDP to be the following function

\[ d_t^{\text{MDP}} : H_t \rightarrow P[A] \]

where \( P[A] \) is a probability distribution over the set of possible actions \( A \). This policy is history dependent as the current and preceding states are all taken into account to decide on the current action. For a Markovian policy, \( H_t = S \) and the decision variables are \( \text{Prob}\{A_t = a_k | X_t = i\} \) for an action \( a_k \in A \) and given any state \( i \).

On the other hand, a decision rule \( d_t \) at time \( t \) is defined as follows for SO-MDPs,

\[ d_t^{\text{SO-MDP}} : H_t \times S \times \{1, \ldots, m\} \rightarrow P[\{\text{Yes, No}\}] \]

where \( H_t \) is the set of all possible state-action history, \( S \) is the set of all possible observed states that can be transitioned to, and \( \{1, \ldots, m\} \) is the set of phases—a phase for each action in the prescribed order. For a Markovian policy, \( H_t = S \) and the decision variables are \( \text{Prob}\{\text{Yes} | X_t = i, O_t = j, Z_t = k\} \), for a state \( i \) and an observed transition to state \( j \) at phase \( k \). Let

\[ \pi = (d_1, d_2, \ldots, d_{N-1}) \]

be the policy for the SO-MDP given that there are \( N-1 \) decision epochs. To show explicit dependence of a probability on the policy, \( \text{Prob}^\pi \{ E \} \) is used to denote the probability of an event \( E \) given a certain policy \( \pi \) (e.g., \( \text{Prob}^\pi \{ A_t = a | X_t = i \} \) is the probability that action \( a \) is taken given that the system is at state \( i \) at time \( t \) and given the policy \( \pi \).

3.3.3 Rewards

Given a state \( s \in S \) and action \( a \in A \), we define the reward \( R_t(s, a) \in \mathbb{R} \subseteq \mathbb{R} \). We define the expected reward for a given decision rule \( d_t \) at time \( t \) to be

\[ v_t(s) = \sum_{k=1}^{m} \text{Prob}^\pi \{ A_t = a_k | X_t = s \} R_t(s, a_k). \]  (2)

Let \( \mathbf{r}_t \in \mathbb{R}^n \) be the vector of expected rewards for each state. Given there are \( N-1 \) decision epochs, then there are \( N \) reward stages and the final stage reward \( \mathbf{r}_N(s) \) is denoted for notation convenience as \( R_N(s, a) = \mathbf{r}_N(s) \) for all \( a \in A \) (or \( \mathbf{r}_N \) the vector having as its elements the final reward at a given state).

3.3.4 State Transitions

We now define the transition probabilities as follows,

\[ G_i(j, k, t) := \text{Prob}\{O_t = j | X_t = i, Z_t = k\}, \]

where \( O_t \) is the observed transition for a given phase. Let \( G_i(t) \in \mathbb{R}^{n \times m} \) be the matrix whose \( j \)-th row and \( k \)-th column entry is \( G_i(j, k, t) \) (for simplicity we drop the index \( t \) when there is no confusion, and transitions are denoted simply by \( G_i \)). Notice that \( G_i \) is independent of the policy \( \pi \) and it models the dynamic environment.

These values are usually obtained from historical data. Let \( G \) be the set containing these transition matrices, \( G_i \).

In standard MDPs, the transition probabilities of the environment are defined in a slightly different way, as \( \text{Prob}\{X_{t+1} = j | X_t = i, A_t = a\} \). In SO-MDPs, contrary to standard MDPs, this latter expression is a function of the policy \( \pi \) as shown later in the proof of Lemma 2.

3.3.5 Sensor Precision/Accuracy

Sensors measuring the transitions may not be perfect. We model measurements for possible transitions as follows:

\[ F_i(l, j) := \text{Prob}\{X_{t+1} = l | X_t = i, O_t = j, \text{Yes} \} \]

where “Yes” corresponds to the decision to accept such a transition. Note that the transition probability is independent from the phase and time at which this observation is obtained. In case of an ideal sensor, \( F_i(l, j) = 1 \) for \( l = j \) and \( F_i(l, j) = 0 \) for \( l \neq j \). The case of sensor failures can be simply modeled with the function \( F \), for example, if a sensor at state \( i \) fails, we have \( F_i(l, j) = 1/n \) for all \( l \in S \). Let \( F \) be the set of all \( F_i \)'s.

3.3.6 Sequentially Observed Markov Decision Processes (SO-MDPs)

A discrete SO-MDP is a 5-tuple \((S, A, G, R, F)\) where \( S \) is a finite set of states, \( A \) is a finite set of actions, \( G \) is the set that contains the transition probabilities as explained above, and \( R \) is the set of rewards.

3.3.7 Performance Metric

The expected total reward is the performance metric, which is given as

\[ v_N^\gamma := \mathbb{E}_x \left[ \sum_{t=1}^{N} \gamma^{t-1} R_t(X_t, A_t) \right], \]  (3)

where \( \gamma \in [0, 1] \) is a discount factor. We consider \( \gamma = 1 \) throughout the paper, but the results remain applicable after scaling the reward function \( R_t \) when \( \gamma < 1 \). \( X_t \) is
the state at decision epoch $t$, $A_t$ is the action due to $\pi$ at decision epoch $t$, and the expectation is conditioned on a probability distribution over the initial states (i.e., $x_1 \in \mathcal{P}[S]$ where $x_1(i) = \text{Prob}\{X_1 = i\}$). When $x_1$ is a point distribution (i.e., there exists a state $s$ such that $x_1(s) = 1$), then we denote the performance metric as $v_N^*(s)$.

3.3.8 Optimal Policy

The optimal policy $\pi^*$ is defined as the policy (when it exists) that maximizes the performance measure, $\pi^* = \arg\max_{\pi} v_N^*$, and $v_N^*$ to be the optimal value, i.e., $v_N^* = \max_{\pi} v_N^*$. For the standard MDP, the backward induction algorithm [19, p. 92] gives the optimal policy as well as the optimal value. However, in the proposed SO-MDPs, the optimization variables are different and a new algorithm is needed to compute the optimal policies. In the following sections, we will provide such an algorithm for SO-MDPs.

3.4 Decision Variables for Markovian Policies

In standard MDPs, the decision variables are the probability distributions for each state $i \in S$, $p_i(a,t) := \text{Prob}\{A_t = a|X_t = i\}$, that define an action selection policy. In the SO-MDPs, the decision variables are whether to accept or reject a given transition at phase $k$, which are defined as follows:

$$P_t(j,k,t) := \text{Prob}\{\text{Accept transition} \mid \text{Observed transition to state} j, \text{System is in state} i, \text{phase} k, \text{and epoch} t\}.$$ 

$$= \text{Prob}\{\text{Yes} \mid O_t = j, X_t = i, Z_t = k\}.$$ 

Since there are $m - 1$ phases, we require $P_t(j,k,m,t) = 1$ so that the final action $a_m$ is taken if all others have been rejected. In standard MDPs, the decision $d_i$ is defined by the decision variables $p_i(a,t) \geq 0$ for all $a \in A$ and decision epoch $t$ satisfying $\sum_a p_i(a,t) = 1$. In the SO-MDP, the decision $d_i$ is defined by the independent matrix variables $\{P_1(t), \ldots, P_m(t)\}$ where $P_i(t) \in \mathbb{R}^{n \times m}$ is the matrix having the probabilities $P_i(j,k,t) \in [0,1]$ for all destination states $j \in S$, for $k = 1, \ldots, m$, and decision epoch $t$. For simplicity we drop the index $t$ when there is no confusion and the variables are denoted simply by $P_i$. Note that this decision rule has a Markovian property because it depends only on the current state.

Next we define an intermediate variable $q_i(a_k)$, which is the probability of choosing action $a_k$ given that the previous actions $a_1, \ldots, a_{k-1}$ are rejected, that is,

$$q_i(a_k) := \text{Prob}\{\text{Yes} \mid X_t = i, Z_t = k\}$$

$$= \sum_{j \in S} \text{Prob}\{\text{Yes}, O_t = j \mid X_t = i, Z_t = k\}$$

$$= \sum_{j \in S} G_i(j,k)P_i(j,k).$$

We observe that $q_i(a_k) = 1$ if $k = m$, i.e., the last action, if reached, is automatically accepted. The probability of choosing action $a_k$, $1 \leq k \leq m$, is the probability that the first $k-1$ actions are rejected (i.e., $\prod_{l=1}^{k-1} (1 - q_i(a_l))$) and then the $k$-th action is accepted (i.e., $q_i(a_k)$):

$$p_i(a_k) := \text{Prob}\{A_t = a_k | X_t = i\}$$

$$= \left(\prod_{l=1}^{k-1} (1 - q_i(a_l))\right) q_i(a_k),$$

where, by convention, $\prod_{l=1}^{0} (1 - q_i(a_l)) := 1$. The above relation shows that the decision variables of the standard MDP $(p_i(a_k)$ for $k = 1, \ldots, m$ and $i = 1, \ldots, n$) are non-convex functions of the decision variables of the SO-MDP ($P_i$ for $i = 1, \ldots, n$).

The transition probability from state $i$ to state $j$ is given by the probability of reaching a phase $k$, accepting the transition to an observed state $i$, and transitioning to state $j$ (due to the imperfect sensing), i.e.,

$$M_i(j,i) := \text{Prob}\{X_{t+1} = j | X_t = i\}$$

$$= \sum_{k=1}^{m} \left(\prod_{l=1}^{k-1} (1 - q_i(a_l))\right) \left(\sum_{l=1}^{n} F_i(j,l)G_i(l,k)P_i(l,k)\right).$$

Note that the above transition probability is not linear in the decision variables of the SO-MDP. Let $x_t(i) = \text{Prob}\{X_t = i\}$ be the probability of being at state $i$ at time $t$, and $x_t \in \mathbb{R}^n$ be the vector of these probabilities. Then the discrete probability distribution evolves according to the following recursive equation, which defines a Markov Chain:

$$x_{t+1} = M_t x_t,$$

where $M_t \in \mathbb{R}^{n \times n}$ (or simply $M$) is the matrix having the elements $M_t(j,i)$ (or simply $M(j,i)$). It is important to note that the $i$-th column of $M$, denoted by $m_i$, is a function of the decision variables in the matrix $P_i$ only (i.e., independent of the variables of the matrices $P_s$ for $s \neq i$).

4 History Dependent and Markovian Policies

In this section we show that, as with standard MDPs [19], it is sufficient to consider only the Markovian policies for SO-MDPs because for any history dependent policy, we can construct a Markovian policy that gives the same total expected reward. First note that the per-
formal metric can be written as
\[ v^*_N(s) = \sum_{t=1}^{N-1} \sum_{j \in S} \sum_{a \in A} R_t(j,a) \text{Prob}^\pi \{ X_t = j, A_t = a \mid X_1 = s \} + \sum_{j \in S} \sum_{a \in A} \tilde{r}_N(j) \text{Prob}^\pi \{ X_N = j, A_N = a \mid X_1 = s \}. \] (8)

Therefore, it is sufficient to show that for any history dependent policy \( \pi \), there exits a Markovian policy \( \pi' \) such that \( \text{Prob}^\pi \{ X_t = j, A_t = a \mid X_1 = s \} = \text{Prob}^{\pi'} \{ X_t = j, A_t = a \mid X_1 = s \} \) for all \( t \). It is indeed the case as the following lemma shows.

Lemma 2 Consider the SO-MDP \((S,A,G,R,F,\gamma)\) with a history dependent policy \( \pi = (d_1,d_2,\ldots) \). Then, for each \( s \in S \), there exists a Markovian policy \( \pi' = (d'_1,d'_2,\ldots) \) satisfying, for \( t = 1,\ldots,N \):
\[ \text{Prob}^\pi \{ X_t = j, A_t = a \mid X_1 = s \} = \text{Prob}^{\pi'} \{ X_t = j, A_t = a \mid X_1 = s \}. \] (9)

PROOF. The proof is given in the Appendix.

Next the above lemma is used to prove the following proposition showing that we can focus, without loss of generality, on Markovian policies.

Proposition 3 Suppose \( \pi \) is a history dependent policy, then for each \( s \in S \) there exists a Markovian policy \( \pi' \) for which the following holds
\[ v^*_N = v'^*_N \quad \text{for} \quad N \geq 1. \]

PROOF. Since the terms on the right-hand side of equation (8) can be replaced by those of a Markovian policy via Lemma 2, we can establish the equivalence of the rewards/costs, i.e., \( v^*_N = v'^*_N \) for \( N \geq 1 \).

5 Dynamic Programming (DP) Approach for SO-MDPs

In this section, we transform the one-dimensional stochastic MDP problem with dynamics (1) into an equivalent \( n \)-dimensional deterministic Dynamic Programming (DP) problem and use this approach to devise an efficient algorithm for finding optimal policies for SO-MDPs. First note that using equation (8), the performance metric can be written as follows:
\[ v^*_N = \sum_{t=1}^{N} x_t^T R_t, \]
where \( x_t \) is the state probability vector propagated as in (7). We can now give the DP formulation. The discrete-time dynamical system describing the evolution of the state probability vector \( x_t \) can then be given by
\[ x_{t+1} = f_t(x_t, P_1(t), \ldots, P_n(t)) = M_t x_t \] (10)
for \( t = 1,\ldots,N-1 \) where \( M_t = M_t(P_1(t),\ldots,P_n(t)) \) is the transition matrix, which is a function of the optimization variables.\(^3\) The above equation shows that the probability vector evolves deterministically. It is important to note that given a policy \( \pi \), the performance metric \( v^*_N \) defined in (3) for the one-dimensional stochastic system (1) is equivalent to the performance of the \( n \)-dimensional deterministic system (10). In other words, any closed-loop feedback law (policy) for the stochastic system defines a policy for the deterministic system with the same performance.

The additive reward per stage is defined as \( g_N(x_N) = x_N^T R_N \) and \( g_t(x_t, P_1(t), \ldots, P_n(t)) = x_t^T R_t \), for \( t = 1,\ldots,N-1 \). Dynamic programming, with a set of admissible controls \( C \), provides a method to calculate the optimal value \( v^*_N \) (and closed-loop policy \( \pi^* \)) via Algorithm 1 [20, Proposition 1.3.1, p. 23].

Algorithm 1 Dynamic Programming
1: Start with \( J_N(x) = g_N(x) \)
2: for \( t = N-1,\ldots,1 \)
\[ J_t(x) = \max_{P_1(t),\ldots,P_n(t) \in C} \left\{ g_t(x, P_1(t), \ldots, P_n(t)) + J_{t+1}(f_t(x, P_1(t), \ldots, P_n(t))) \right\}. \]
3: Result: \( v^*_N = J_1(x_1) \) where \( x_1 \) is the initial state probability distribution.

5.1 Backward Induction for the SO-MDP model

This section presents the backward induction algorithm for solving the SO-MDP by using the dynamic programming approach. The set of admissible controls at time \( t \) is given by \( C \) defined as follows:
\[ 0 \leq P_i(j,k,t) \leq 1 \quad \text{for all} \quad i \in S, j \in S, a_k \in A. \]

Using Algorithm 1, we can now give the following proposition:

Proposition 4 The term \( J_t(x) \) in the dynamic programming algorithm for the SO-MDP has the following closed-form solution:
\[ J_t(x) = x^T V_t^*, \]
\(^3\) Note that \( i \)-th column of \( M_t \) is a function of only \( P_i(t) \) matrix.
where $V_t^*\vDash$ is a vector that satisfies the following recursion $V_N^* = \mathbf{r}_N$, and for $t = N - 1, \ldots, 1$ we have

$$V_t^*(i) = \max_{P_t(i)} \left\{ \bar{r}_t(i) + \sum_{j,s} M_t(j,i) V_{t+1}^*(j) \right\}$$

for $i = 1, \ldots, n$, where $M_t(i,t)$ is the $i$-th column of $M_t$.

**PROOF.** The proof is given in the Appendix.

Notice that $J_t(x)$ has a closed-form equation as a function of $x$ and so it suffices to calculate $V_t^*$ for $t = N, \ldots, 1$ for finding the optimal value of the SO-MDP given by $v_N^* = J_1(x_1) = x_1^T V_1^*$. The backward induction algorithm is given in Algorithm 2.

**Algorithm 2** Backward Induction: Sequential MDP Optimal Policy

1. Start with $V_N^*(i) = \bar{r}_N(i)$ for all $i \in S$
2. for $t = N - 1, \ldots, 1$ given $V_{t+1}^*$ and for $i = 1, \ldots, n$ calculate the optimal value

$$V_t^*(i) = \max_{P_t(i)} \left\{ \bar{r}_t(i) + \sum_{j \in S} M_t(j,i) V_{t+1}^*(j) \right\}$$

and the optimal policy $P_t(i,t)$ given by:

$$P_t^*(i,t) = \arg\max_{P_t(i)} \left\{ \bar{r}_t(i) + \sum_{j \in S} M_t(j,i) V_{t+1}^*(j) \right\}$$

3. Result: $v_N^* = x_1^T V_1^*$ where $x_1$ is the initial state probability distribution.

**Remark 5** We want to stress two points about the algorithm. First, the policy calculated by Algorithm 2 is optimal (maximizing the total expected reward) because of line 3 in Algorithm 1 and Proposition 4. Second, $\bar{r}_t(i)$ and $M_t(j,i)$ are both functions of the decision variables in $P_t$. In standard MDPs, these values are simply linear in the decision variables. However, in SO-MDP, these values are non-convex in the decision variables and further analysis is needed for a numerically tractable implementation of the algorithm.

**5.2 A Numerically Tractable Implementation of Algorithm 2**

At each iteration of Algorithm 2, the value function at a given decision epoch $t$ is given by the following equation:

$$V_t^*(i) = \max_{P_t(i)} \bar{r}_t(i),$$

where $V_t(i) = \bar{r}_t(i) + \sum_{j \in S} M_t(j,i) V_{t+1}^*(j)$. In this formulation, $\bar{r}_t(i)$ and $M_t(j,i)$ are functions of the decision variable $P_t(i)$, for given state $i$ and time epoch $t$, where explicit expressions can be deduced from Eq. (2), Eq. (5), and Eq. (6) as follows:

$$\bar{r}_t(i) = \sum_{a \in A} p_t(a) R_t(i, a)$$

$$M_t(j,i) = \sum_{k=1}^{m} \left[ \prod_{l=1}^{k-1} (1 - q_t(a_l)) \right] q_t(a_k) R_t(i, a_k)$$

where $q_t(a_k) = \sum_j G_t(j,k) P_t(j,k)$. By substituting these equations into the expression of $V_t(i)$, we obtain

$$V_t(i) = \bar{r}_t(i) + \sum_{j \in S} M_t(j,i) V_{t+1}^*(j)$$

$$= \sum_{k=1}^{m} \left[ \prod_{l=1}^{k-1} (1 - q_t(a_l)) \right] \left[ \sum_{j=1}^{n} G_t(j,k) P_t(j,k) \right] R_t(i, a_k)$$

$$+ \sum_{j=1}^{n} \left[ \sum_{k=1}^{m} \left[ \prod_{l=1}^{k-1} (1 - q_t(a_l)) \right] \left[ \sum_{l=1}^{n} F_t(j,l) G_t(l,k) P_t(l,k) \right] V_{t+1}^*(j) \right]$$

where

$$Y_t(i,k) := G_t(i,k) P_t(i,k) \prod_{l=1}^{k-1} (1 - q_t(a_l))$$

$$H_t := 1_n e^T R_t + F^T V_{t+1}^* \mathbf{1}_m^T$$

$e^T R_t$ is the $i$-th row of the reward matrix $R_t$, and $1_n$ (alternatively $\mathbf{1}_m$) is the column vector of all ones with $n$ (alternatively $m$) elements. Note that $H_t$ is independent of the decision variable $P_t$, and $V_t(i)$ is linear in $Y_t$.

For efficient implementation of the algorithm, it remains to show what conditions $Y_t(i,k)$ should satisfy so that the mapping $Y_t(i,k) = \prod_{l=1}^{k-1} (1 - q_t(a_l)) P_t(j,k) G_t(i,k)$ is invertible. Notice that if $q_t(a_l) \neq 1$ for $l = 1, \ldots, m-1$, then the mapping is one-to-one, and we will give the expression for $P_t$ in terms of $Y_t$ shortly after. If there exists $l$ such that $q_t(a_l) = 1$, then the phases $k > l_{\text{min}}$ are not
reached because an earlier action must necessarily be accepted where \( l_{\text{min}} = \min\{l | g_i(a_l) = 1 \} \). This means that \( V_i(i) \) is independent of \( P_i(j, k) \) when \( k > l_{\text{min}} \) (i.e., the optimal value is not affected by these variables), and without loss of generality we can consider \( P_i(j, k) = 1 \) for \( j = 1, \ldots, n \) and \( k = l_{\text{min}} + 1, \ldots, m \).

We can give now the expression of \( P_i \) in terms of \( Y_i \) by the following lemma:

**Lemma 6** For a given state \( i \), the following equation holds for \( Y_i(j, k), j = 1, \ldots, n \) and \( k = 1, \ldots, m \), in Eq. (16):

\[
Y_i(j, k) = \left( 1 - \sum_{l=1}^{k-1} \sum_{s=1}^n Y_i(s, l) \right) P_i(j, k) G_i(j, k).
\]

**(PROOF.** The proof is given in the Appendix.

It remains to derive the constraints on \( Y_i(j, k) \) when \( P_i \in C \). Since \( P_i(j, k) \in [0, 1] \) for all \( j = 1, \ldots, n \) and \( k = 1, \ldots, m - 1 \), we obtain the following conditions:

\[
0 \leq Y_i(j, k) \leq \left( 1 - \sum_{l=1}^{k-1} \sum_{s=1}^n Y_i(s, l) \right) G_i(j, k),
\]

and since by definition \( P_i(j, m) = 1 \) for all \( j = 1, \ldots, n \) because \( m \) is the index for the last control action in the sequence and the last action is always accepted, then from Lemma 6 we can write

\[
Y_i(j, m) = \left( 1 - \sum_{l=1}^{m-1} \sum_{s=1}^n Y_i(s, l) \right) G_i(j, m).
\]

As a result, \( V^*_i(i) \) is the solution of the following linear program

\[
\begin{align*}
\text{maximize} & \quad \text{Tr}(H_i^T Y_i) \\
\text{subject to} & \quad \text{for } j = 1, \ldots, n \text{ and } k = 1, \ldots, m - 1 \\
& \quad 0 \leq Y_i(j, k) \leq \left( 1 - \sum_{l=1}^{k-1} \sum_{s=1}^n Y_i(s, l) \right) G_i(j, k) \\
& \quad Y_i(j, m) = \left( 1 - \sum_{l=1}^{m-1} \sum_{s=1}^n Y_i(s, l) \right) G_i(j, m)
\end{align*}
\]

To write it in matrix form, let \( 1_n \) be the column vector of all ones and dimension \( n, J = 1_n 1_n^T \), and \( B \) be a constant \( m \times m \) matrix defined as \( B(l, k) = 1 \) if \( k > l \) and \( B(l, k) = 0 \) otherwise. The linear program for finding the optimal value and policy at a given internal for-loop iteration of Algorithm 2 is given as follows:

\[
\begin{align*}
\text{maximize} & \quad \text{Tr}(H_i^T Y_i) \\
\text{subject to} & \quad Y_i + (JY_i B) \odot G_i \leq G_i \\
& \quad (Y_i + (JY_i B) \odot (G_i - G_i)) e_m = 0 \\
& \quad Y_i \geq 0
\end{align*}
\]

Where \( \odot \) is the elementwise, or Hadamard, product and \( \leq \) is the elementwise inequality. Let \( Z_i(j, k) = \left( 1 - \sum_{l=1}^{k-1} \sum_{s=1}^n Y^*_i(s, l) \right) G_i(j, k) \) if \( k = 2, \ldots, m \) and \( Z_i(j, 1) = 1 \) for all \( i, j = 1, \ldots, n \). The following proposition summarizes our results:

**Proposition 7** For a given decision epoch \( t \) and state \( i \), the optimal value and optimal policy due to Algorithm 2 are given by

\[
V^*_i(i) = \text{Tr}(H_i^T Y^*_i),
\]

and for \( j = 1, \ldots, n \) and \( k = 1, \ldots, m \)

\[
P^*_i(j, k, t) = \begin{cases} 
Y^*_i(j, k)/Z_i(j, k) & \text{if } Z_i(j, k) > 0, \\
1 & \text{else}.
\end{cases}
\]

where \( Y^*_i \) is the solution of the linear program (18).

**(PROOF.** The proof is based on the fact that the linear program over the decision variables \( Y_i \) is equivalent to the original optimization over the \( P_i \) variables when considering the additional (redundant) constraints: \( P_i(j, k) = 1 \) for \( j = 1, \ldots, n \) and \( k = l_{\text{min}} + 1, \ldots, m \).

### 6 Deterministic Policies: Extending State Space

In Section 4 we have shown that optimal policies are Markovian. We then provided a linear program that computes an optimal policy. In this section, we will show that there exists a deterministic optimal policy \( \pi^* \) that maximizes \( v^*_N \) with \( P^*_i(j, k) \in \{0, 1\} \). We will also provide a dynamic programming algorithm for the optimal deterministic policy.

To show that deterministic policies are sufficient for optimality, we will extend the state space of the system to include the current state, sensor measurement, and phase: let \( S = S \times S \times \{1, \ldots, m\} \) be the extended state of the system. Fig. 3 shows the transition structure of the system. Each node in the figure corresponds to a state \((X_t, O_t, Z_t) \in S \). It is important to note that the extended system is not a standard MDP with augmented state space because there exist internal transitions within a decision epoch. However, the connected state has a special structure: it is a directed acyclic graph (DAG). This structure allows efficient algorithms to run on the network by ordering the states [21]. We will use
the DAG property to devise an efficient dynamic programming algorithm to calculate the performance $v^*_N$ given any Markovian policy $\pi$. We associate with every node in the network of Fig. 3 a scalar value:

$$f^{\pi}(t, k, i, j) := \mathbb{E}^{\pi} \left[ \sum_{u=t}^{N} R(X_u, A_u)|X_u = i, Z_u = k, O_u = j \right]$$

Given the structure of the network and the SO-MDP model description, $f^{\pi}(t, k, i, j)$ satisfies a recursion defined as follows: if the observed transition is accepted, then the total expected reward is equal to the immediate collected reward $R(i, k)$ plus the expected value of the scalars associated with nodes in the $t+1$-st decision epoch block reachable from $(t, k, i, j)$. If the observed transition is rejected, then the total expected reward is equal to the expected value of the scalars associated to nodes in the $t$-th decision epoch block reachable from $(t, k, i, j)$. Therefore, $f^{\pi}(t, k, i, j)$ satisfies the following equation:

$$f^{\pi}(t, k, i, j) = P_t(j, k, t)g^{\pi}(t, k, i, j) + (1 - P_t(j, k, t))h^{\pi}(t, k, i, j) \quad \text{(20)}$$

where

$$g^{\pi}(t, k, i, j) = R(i, k) + \sum_{l=1}^{n} F_l(j, k) \sum_{v=1}^{n} G_l(v, 1)f^{\pi}(t + 1, 1, l, v) \quad \text{(21)}$$

$$h^{\pi}(t, k, i, j) = \sum_{v=1}^{n} G_l(v, k + 1)f^{\pi}(t, k + 1, i, v) \quad \text{(22)}$$

with a boundary condition

$$f^{\pi}(N, 1, i, j) = \bar{r}_N(i) \text{ for } i, j = 1, \ldots, n. \quad \text{(23)}$$

Then the performance of a policy can then be given as follows:

$$v^*_N = \sum_{i,j=1}^{n} x_1(i)G_i(j, 1)f^{\pi}(1, 1, i, j). \quad \text{(24)}$$

For any given policy, $v^*_N$ can be computed efficiently using the dynamic programming equation (20) and the terminal condition (23). The following proposition shows that deterministic policies can be constructed to replace randomized policies.

**Proposition 8** For any randomized policy $\pi_r$, there exists a deterministic policy $\pi_d$ such that $v^*_{\pi_d} \geq v^*_{\pi_r}$.

**PROOF.** First note that a deterministic policy means that $P_t(j, k, t) \in \{0, 1\}$ for all $i, j, k,$ and $t$. The proof proceeds by constructing a deterministic policy from the randomized policy with a better total expected reward. Let $\pi_r$ be a randomized policy, then there exist $i, j, k,$ and $t$ such that $P^r_t(j, k, t) \in (0, 1)$. Let $\pi$ be the same policy as $\pi_r$ except for

$$P^\pi_t(j, k, t) = \begin{cases} 1 & \text{if } g^\pi_t(t, k, i, j) \geq h^\pi_t(t, k, i, j) \\ 0 & \text{else}. \end{cases}$$

With this modified policy $\pi$, it is easy to check that $v^*_N \geq v^*_\pi$ because $f^\pi(t, k, i, j) \geq f^\pi_t(t, k, i, j)$ and all the coefficients of $f^\pi$’s in $v^*_N$ are positive (probabilities). We can repeat this process until there exist no $i, j, k,$ and $t$ such that $P^\pi_t(j, k, t) \in (0, 1)$. The resulting policy $\pi = \pi_d$ with $v^*_{\pi_d} \geq v^*_N$.

A deterministic optimal policy, ensured by Proposition 8, can be calculated using the dynamic programming equation (20) as in Algorithm 3.

**Remark 9** Algorithm 2 and Algorithm 3 both provide an optimal policy for the SO-MDP model. These two approaches are different, each having advantages and disadvantages. Algorithm 2 works directly with the state space $S$ without the need to expand it and is based on linear programming. The LP approach gives the possibility of adding constraints to the MDP problem, e.g., bounding the probability of choosing an action $p_t(a_k)$ or adding safety constraints on the state probability distribution $x_t$ [22]. The resulting policy from Algorithm 2 can be stochastic. Algorithm 3 on the other hand works with an extended state space $\bar{S}$, but is faster in practice and more numerically stable because the algorithm heavily exploits the structure of the problem. The disadvantage, however,
Algorithm 3 An Optimal Deterministic Policy for SO-MDP

1: Start with \( f^*(N, 1, i, j) = \tilde{r}_N(i) \) for \( i, j = 1, \ldots, n \), and \( h^*(t, m, i, j) = \min_{i,k} R_t(i, k) \).

2: for \( t = N - 1, \ldots, 1 \)
   for \( k = m, \ldots, 1 \)
   for all \( i, j = 1, \ldots, n \)
   \[
   f^*(t, k, i, j) = \max\{g^*(t, k, i, j), h^*(t, k, i, j)\}
   \]
   where \( g(.) \) and \( h(.) \) are defined as in (21) and (22) respectively. The optimal decision
   \[
   P^*_t(i, k, t) = \begin{cases} 
   1 & \text{if } g^*(t, k, i, j) \geq h^*(t, k, i, j) \\
   0 & \text{else.}
   \end{cases}
   \]

3: Result: \( v^*_N = \sum_{i,j=1}^{n} x_1(i) G_1(j, 1) f^*(1, 1, i, j), \) and \( V^*_t(i) = \sum_{j=1}^{n} G_t(j, 1) f^*(t, 1, i, j) \) for all \( i \) and \( t \) where \( V^*_t(i) \) is given in (11).

is that the algorithm is not flexible in handling additional constraints in the decision problem. The resulting policy from Algorithm 3 is deterministic by construction.

7 Complexity of SO-MDP

Algorithm 3 can be implemented efficiently. The memory requirement for the algorithm is \( O(Nmn^2) \) required to represent the MDP system parameters \((S, A, G, R, F, \gamma)\), the policy \( \pi \), and the scalars \( f^* \). An efficient implementation of Algorithm 3 requires only \( O(Nmn^2) \) computational complexity because in line two of the algorithm we have \( N - 1 \) decision epochs, \( m \) phases, and the operation \( \max\{g^*, h^*\} \) can be computed in \( O(m) \) because the expression of \( h \) in (22) is independent of \( j \) and the inner summation in the expression of \( g \) in (21) is independent of \( i \) so they can be computed in an outer loop. In case of perfect sensors, i.e., \( F_i = I \) for all \( i \), \( \max\{g^*, h^*\} \) can be computed in \( O(1) \) and thus the overall complexity would reduce to \( O(Nmn^2) \), which turns out to be the same complexity encountered in the standard MDPs.

8 Order of Actions of SO-MDP

In some applications, the order of the actions to be used is fixed by the physical limitation of the system. In the general case, with \( N \) decision epochs, there are \((m!)^N\) possible orderings for actions, and finding the optimal ordering is not computationally feasible. We provide here a heuristic on the order of actions that would maximize a lower bound on the performance metric.

Note that any feasible policy for a standard MDP is also a feasible (but not optimal) solution for the proposed sequential model (i.e., \( \pi_{\text{MDP}} \subseteq \pi_{\text{SOMDP}} \)), then the following holds for any given ordering of actions:

\[
\pi^*_t \geq v^*_t, \text{ for all standard MDP policies } \pi_{\text{MDP}}.
\]

Therefore we can write:

\[
\mathbb{E} \pi^*_{\text{SOMDP}} \left[ \sum_{u=t}^{N} R(X_u, A_u) | X_u = i, Z_u = k, O_u = j \right] \geq \mathbb{E} \pi_{\text{MDP}} \left[ \sum_{u=t}^{N} R(X_u, A_u) | X_u = i, A_u = a_k \right]
\]

(25)

The left-hand-side of (25) is simply \( f^*_{\text{SOMDP}}(t, k, i, j) \) by definition, and the right-hand-side denoted by \( Q_t(i, a) \) is known as the optimal state-action value function of the MDP, or simply Q-function. As a result, to maximize a lower bound on the performance metric \( v^*_{\text{SOMDP}} \), the action to be chosen in the first order (first phase) is the action that maximizes the Q-function of the standard MDP, i.e.,

\[
a_1 = \arg \max_{a \in A} Q_t(i, a).
\]

(26)

Using a similar argument for the proceeding phases, the action to be selected would be the action that maximizes \( Q_t(i, a) \) other than the actions that have already been selected. In summary, for a given decision epoch \( t \) and state \( i \), we sort the actions in decreasing order of their \( Q \) values. This order of actions would then be used in the SO-MDP formulation. Simulations show that indeed this heuristic performs better than a fixed or random ordering of actions.

9 Simulations

This section compares the traditional MDP model to our SO-MDP model through numerical simulations. An autonomous robot is tasked to explore a square region which is divided into many subregions (or bins). There are five possible action for the robot to take at each time epoch: move up, move down, move left, move right, or stay still. If the agent is on the boundary, it may transition to any adjacent cell deterministically. Otherwise, the agent moves in the desired direction with 0.6 probability, or in one of the four other directions, each with 0.1 probability. The rewards for taking action \( a_k \) at state \( i \) and the terminal rewards at each state are randomly generated from a uniform distribution on the interval \([0, 100]\). The objective is to maximize the undiscounted total reward over \( N = 20 \) decision epochs.

The standard MDP chooses action at the start of each decision epoch. SO-MDP, on the other hand, has the choice to accept or reject each transition after observing the potential result of choosing each action in the order “move up”, “move down”, “move left”, “move right”, and “stay still”. Figure 4 shows that in the finite horizon
Timing comparison for the dynamic programming methods for regular MDP and SO-MDP, and the linear programming method for SO-MDP \((n = (\text{Grid Size})^2, m = 5, N = 20)\). Computation time of 1000 seconds is selected as the limit for the solver, for Grid Size greater than 16 \((n \geq 256 \text{ states})\), the LP solver could not find a solution in less than 1000 seconds.

Expected reward given a uniformly distributed initial state for the regular MDP and SO-MDP. In this case, dynamic programming was much faster than the linear programming algorithm. Note that the number of states increases quadratically with the grid size, and the number of actions remains constant. The SO-MDP policy took up to an order of magnitude as much time to compute than for the regular MDP, but both were much faster than CVX for MATLAB with the solver Mosek.

Figure 5 shows the increase in the total expected reward from sequential observation. The benefit of observing state transitions increases with grid size for this example, though at a decreasing rate. These results suggest that SO-MDP has a reward advantage over standard MDP which tends to increase with the problem size and computational effort, and so SO-MDP is most suitable for problems whose reward increase is greater than any extra costs from the observation process or from generating an optimal policy.

A random set of rewards was generated over a 50 × 50 grid as before. The left side of Figure 6 shows that by allowing state transitions to be observed, the total reward is increased everywhere there is nondeterminism in the dynamics, i.e. all states not on the boundary. To model a set of states with high uncertainty, a low-reward region was added in a 5 × 5 square centered at coordinates (10, 15) to model an obstacle, and with -10 reward for each action. The right side of Figure 6 shows that the advantage of SO-MDP over MDP is most evident near this obstacle, since the agent is better able to move away from that area quickly by being able to reject transitions to unfavorable states before they would occur.

In some situations, it may be possible for an agent to choose the order in which to observe the potential state transitions. Another set of simulations were performed to demonstrate the effect of a different observation order. The left side of Figure 7 shows the value function under SO-MDP with the action order of up, down, left, right, stay. The observation order was then modified to be a function of the state and time step according to the heuristic that actions should be observed in order of decreasing Q values. The difference in value between the flexible and fixed observation orders is shown in the right side of Figure 7. This additional freedom increased the expected total reward on average, especially above the high-cost region, although some states did not benefit from the new ordering of observations.

Conclusion

This paper introduces SO-MDP model, a new model for MDPs that incorporates the additional observations of state transitions for a given action, in a sequential manner. This model achieves better expected total rewards than the optimal policies for the standard MDP models studied in the literature due to the utilization of additional sensed information. We showed that optimal policies for SO-MDP are Markovian and deterministic. We accordingly devised two efficient algorithms to find
an optimal SO-MDP policy: the first is based on linear programming that allows for additional constraints on the model, and another algorithm based on dynamic programming with similar computational complexity to standard MDP policy synthesis algorithms.

Appendix

A Proof of Lemma 2

Let \( \pi' \) be a Markovian policy. Then the decision variables at a decision epoch \( t \) are defined as the matrices \( \{P_i(t), i = 1, \ldots, n\} \). The proof proceeds by constructing these decision variables such that the following equations hold for all \( t \):

\[
\text{Prob}\left\{ X_t = j \mid X_{t-1} = i, A_{t-1} = a_k \right\} = \frac{\text{Prob}\left\{ X_t = j \mid X_{t-1} = i, A_{t-1} = a_k, X_1 = s \right\}}{G_i(j, k, t) \prod_{l=1}^{k-1} (1 - q_i(l))}, \tag{A.1}
\]

\[
\text{Prob}\left\{ X_t = i \mid A_{t-1} = a_k \right\} = \text{Prob}\left\{ X_t = i \mid A_{t-1} = a_k, X_1 = s \right\}. \tag{A.2}
\]

For simplicity, we consider perfect sensors, i.e., \( F_i = I \) for all \( i \), but the proof can be easily extended for the general case. Define the following Markovian policy \( d'_t \) for a nonzero denominator where the decision variables \( P_i(j, k, t) \) are chosen as function of \( \pi \) as follows:

\[
P_i(j, k, t) = \frac{\text{Prob}\left\{ X_{t+1} = j, A_t = a_k \mid X_t = i, X_1 = s \right\}}{G_i(j, k, t) \prod_{l=1}^{k-1} (1 - q_i(l))}. \tag{A.3}
\]

Without loss of generality, we can choose \( P_i(j, k, t) = 1 \) when \( G_i(j, k, t) = 0 \) or when \( \prod_{l=1}^{k-1} (1 - q_i(l)) = 0 \). Next we show that the numerator is always less than or equal to the denominator to have a well-defined \( P_i(j, k, t) \).

Let \( L = \{\sigma \in H_t \mid X_1 = s, X_t = i\} \), \( \alpha_\sigma = \text{Prob}\left\{ \sigma \right\} \), \( P^*_t(j, k, t) \) is the history dependent decision variable, \( q_i^*(a_k) = \sum_{s \in S} G_i(s, k, t) P^*_t(s, k, t) \), and

\[
\alpha_{\sigma, k} = \alpha_\sigma \left( \prod_{l=1}^{k-1} (1 - q_i^*(a_l)) \right). \tag{A.4}
\]

It can be shown by induction on \( k \) that \( P_i(j, k, t) = \sum_{\alpha_{\sigma, k}} \frac{\alpha_{\sigma, k} P^*_t(j, k, t)}{\sum_{\alpha_{\sigma, k}}} \). Thus \( P_i(j, k, t) \) is a well-defined probability. Let us show next that equation (A.1) holds.

\[
\text{Prob}\left\{ X_t = j \mid X_{t-1} = i, A_{t-1} = a_k \right\} = \frac{\text{Prob}\left\{ X_t = j \mid X_{t-1} = i, A_{t-1} = a_k, X_1 = s \right\}}{G_i(j, k, t) \prod_{l=1}^{k-1} (1 - q_i(l))}.
\]

Let us show now equation (A.2).

\[
\text{Prob}\left\{ X_t = a_k \mid X_{t-1} = i \right\} = p_i(a_k) = \prod_{l=1}^{k-1} (1 - q_i(a_l)) q_i(a_k)
\]

\[
= \sum_i \left( \prod_{l=1}^{k-1} (1 - q_i(a_l)) \right) P_i(j, k, t) G_i(j, k, t)
\]

Now by using Equation (A.3),

\[
p_i(a_k) = \sum_{j=1}^{n} \text{Prob}\left\{ X_{t+1} = j, A_t = a_k \mid X_t = i, X_1 = s \right\}
\]

\[
= \text{Prob}\left\{ A_t = a_k \mid X_t = i, X_1 = s \right\}.
\]

Now that we have established equations (A.1) and (A.2), we can proceed by using induction to complete the proof of the lemma as done for the standard MDPs in [19, Theorem 5.5.1]. Note that, in the standard case, establishing (A.1) is trivial, which is not the case for SO-MDPs.

Clearly equation (9) holds with \( t = 1 \). Assume it holds also for \( t = 2, \ldots, u - 1 \). Then

\[
\text{Prob}\left\{ X_u = j \mid X_1 = s \right\} = \sum_{i=1}^{n} \sum_{k=1}^{m} \left( \text{Prob}\left\{ X_{u-1} = i, A_{u-1} = a_k \mid X_1 = s \right\} \times \text{Prob}\left\{ X_u = j \mid X_{u-1} = i, A_{u-1} = a_k \mid X_1 = s \right\} \right)
\]

\[
= \sum_{i=1}^{n} \sum_{k=1}^{m} \left( \text{Prob}\left\{ X_{u-1} = j, A_{u-1} = a_k \mid X_1 = s \right\} \times \text{Prob}\left\{ X_u = j \mid X_{u-1} = j, A_{u-1} = a_k \mid X_1 = s \right\} \right)
\]

\[
= \text{Prob}\left\{ X_u = j \mid X_1 = s \right\},
\]

where we used the induction hypothesis and (A.1). Therefore,

\[
\text{Prob}\left\{ X_u = j, A_u = a_k \mid X_1 = s \right\} = \text{Prob}\left\{ X_u = a_k \mid X_1 = j \right\} \text{Prob}\left\{ X_u = j \mid X_1 = s \right\} \text{Prob}\left\{ X_u = j \mid X_1 = s \right\}
\]

\[
= \text{Prob}\left\{ X_u = j, A_u = a_k \mid X_1 = s \right\}
\]

where we used equations (A.2) and (A.4). This ends the proof.

B Proof of Proposition 4

We will show this by induction. From the definition of \( g_N(x) \) we have the base case satisfied, i.e., \( J_0(x) = x'F_N = x'V^*_N \). Supposing that the hypothesis is true
from $N - 1, \ldots, t + 1$, we then show it is true for $t$. From the DP algorithm, we can write

$$J_t(x) = \max_{P_t(1), \ldots, P_t(T) \in C} \left\{ x^T \bar{r}_t + J_{t+1}(M_t x) \right\} \quad (B.1)$$

$$= \max_{P_t(1), \ldots, P_t(T) \in C} \left\{ x^T \bar{r}_t + x^T M_t^T V_{t+1}^* \right\} \quad (B.2)$$

$$= \max_{P_t(1), \ldots, P_t(T) \in C} \left\{ \sum x_i (\bar{r}_i(i) + m_{i,T}^T V_{t+1}^*) \right\} \quad (B.3)$$

$$= \sum x_i \left( \max_{P_t(1), \ldots, P_t(T) \in C} \left\{ \bar{r}_i(i) + m_{i,T}^T V_{t+1}^* \right\} \right) \quad (B.4)$$

where $m_{i,T}^T$ indicates the transpose of the $i$-th column of $M_t$ which is a function of the decision variables of the $P_t$ matrix only. The transition from (B.1) to (B.2) is from the induction assumption, and the transition from (B.3) to (B.4) is because $x_i \geq 0$ for all $i$ and the function is separable in terms of the optimization variables. The maximization inside the parentheses is nothing but $V_t^*(i)$, so then $J_t(x) = \sum x_i V_t^*(i) = x^T V_t^*$. This ends the proof.

C Proof of Lemma 6

We will prove this lemma by showing $\Pi_{t=1}^{k-1} (1 - q_t(a_t)) = 1 - \sum_{i=1}^{k-1} a_{i=1}^n Y_i(s, l)$ by induction. It is true for $k = 2$ because by definition $q_t(a_t) = \sum_{s \in G} P_t(s, l) = \sum_{s=1}^n Y_i(s, l)$. Suppose it is true till $k - 2$, and let us show it true for $k - 1$. We have

$$\Pi_{t=1}^{k-1} (1 - q_t(a_t)) = \left( \Pi_{t=1}^{k-2} (1 - q_t(a_t)) \right) (1 - q_{k-1}(a_{k-1}))$$

$$= \left( \Pi_{t=1}^{k-2} (1 - q_t(a_t)) \right) - \sum_{s=1}^n Y_i(s, k - 1)$$

$$= 1 - \sum_{l=1}^{k-1} \sum_{s=1}^n Y_i(s, l),$$

where the last equality uses the induction hypothesis.

References


