The Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-23)

Planning and Learning for Non-markovian Negative Side Effects Using Finite State Controllers

Aishwarya Srivastava[1], **Sandhya Saisubramanian**[2], **Praveen Paruchuri**[1],
Akshat Kumar[3], **Shlomo Zilberstein**[4]
[1]IIIT Hyderabad
[2]Oregon State University
[3]Singapore Management University
[4]University of Massachusetts Amherst
aishwarya.srivastava@research.iiit.ac.in, sandhya.sai@oregonstate.edu, praveen.p@iiit.ac.in,
akshatkumar@smu.edu.sg, shlomo@cs.umass.edu

Abstract

Autonomous systems are often deployed in the open world where it is hard to obtain complete specifications of objectives and constraints. Operating based on an incomplete model can produce *negative side effects* (NSEs), which affect the safety and reliability of the system. We focus on mitigating NSEs in environments modeled as Markov decision processes (MDPs). *First*, we learn a model of NSEs using observed data that contains state-action trajectories and severity of associated NSEs. Unlike previous works that associate NSEs with state-action pairs, our framework associates NSEs with entire trajectories, which is more general and captures *non-Markovian* dependence on states and actions. *Second*, we learn finite state controllers (FSCs) that predict NSE severity for a given trajectory and generalize well to unseen data. *Finally*, we develop a constrained MDP model that uses information from the underlying MDP and the learned FSC for planning while avoiding NSEs. Our empirical evaluation demonstrates the effectiveness of our approach in learning and mitigating Markovian and non-Markovian NSEs.

Introduction

As autonomous systems are increasingly deployed in open-world environments, obtaining perfect description of the target environment becomes practically infeasible (Dietterich 2017). Model incompleteness may arise in the form of underspecified objectives or missing constraints due to the limited availability of information or unintentional overlooking of details, especially those considered unrelated to the agent's primary task during system design (Saisubramanian, Zilberstein, and Kamar 2022). Operating based on such models may produce *negative side effects* (NSEs)—which are unintended, undesired consequences of agent actions that occur in addition to the intended effects, often discovered after deployment (Amodei et al. 2016; Alizadeh Alamdari et al. 2022; Krakovna et al. 2019; Saisubramanian, Roberts, and Zilberstein 2021).

Addressing NSEs is gaining increased attention since it affects the safety and reliability of deployed AI systems. The NSEs may be Markovian or non-Markovian, depending on the problem setting (Saisubramanian, Kamar, and Zilberstein 2022). Markovian NSEs are those associated with the immediate execution of an action in a state. Non-Markovian NSEs are associated with a sequence of actions. Many real-world domains are characterized by non-Markovian NSEs.

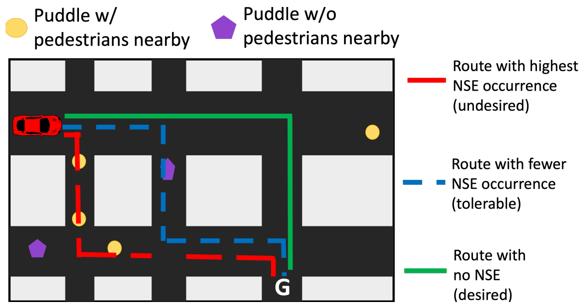

Puddle w/ pedestrians nearby
Puddle w/o pedestrians nearby
Route with highest NSE occurrence (undesired)
Route with fewer NSE occurrence (tolerable)
Route with no NSE (desired)
G

Figure 1: Non-Markovian NSE in a driving domain, NSE is associated with frequency of driving fast through puddles.

For example, consider an autonomous vehicle (AV) that aims to quickly navigate to a goal location while complying with the traffic rules (Figure 1). While the AV's model may include all the details relevant to this task, it may lack superfluous details, such as the impact of driving fast through puddles. Based on this model, the AV may drive fast through puddles, producing an NSE. While the user may tolerate this behavior occasionally, they may not be willing to tolerate the AV frequently splashing water on nearby pedestrians. Since the model has no information about NSE, the AV's state representation may not include a feature indicating the number of times it drove fast through puddles. Thus the severity of NSE occurrence in this case depends on the agent's trajectory and is therefore *non-Markovian* for its state representation, which is Markovian for the navigation task.

Prior works mitigate Markovian NSE through model and policy updates (Saisubramanian, Kamar, and Zilberstein 2020), by constraining actions (Zhang, Durfee, and Singh 2018, 2020), by minimizing deviations from a baseline (Krakovna et al. 2019), by incentivizing the agent to preserve the ability to perform future tasks in the environment (Krakovna et al. 2020), and by considering the influence of actions on other agents in the environment (Alizadeh Alamdari et al. 2022). In the real world, however, NSE may be associated with a trajectory and the state representation may not be sufficient for the NSE to be Markovian, particularly when the features determining NSEs are unrelated to the agent's primary task. It is not straightforward to extend the existing approaches to mitigate non-Markovian NSEs since the NSE penalty for a trajectory may not be decomposable into additive penalties associated with each state-action pair (Saisubramanian, Kamar, and Zilberstein 2022). In addition, it is often computationally expensive to expand the state representation for the NSEs to be Markovian, as the expanded representation may make the primary planning task intractable.

We propose controller-assisted safe planning (CASP), a paradigm to mitigate the impact of *non-Markovian NSEs*—the state representation is assumed to be *Markovian* for the agent's primary task but the NSEs may be non-Markovian for this representation. The problem is formulated as a *constrained Markov decision process* (CMDP), with constraints on NSE occurrences and deviation from the initial objective value, denoted by the slack. The slack indicates the maximum allowed deviation from the optimal expected value when the agent updates its policy to mitigate NSEs. Since the agent has no prior knowledge about NSEs, it must learn a predictive model of NSEs which is then used to avoid them. Specifically, our approach uses a three-step method to detect and mitigate NSEs (Figure 2): (1) the agent gathers information about the side effects of its trajectories through oracle (typically human) feedback; (2) the gathered information is used to learn a predictive model of (non-Markovian) NSEs, using a finite state controller (FSC) and the EM algorithm (Dempster, Laird, and Rubin 1977); and (3) the agent replans to mitigate NSEs using the learned model, given a tolerance threshold and a slack, by solving a CMDP using a variant of the standard dual linear program for MDPs (Puterman 1990) to factor the learned FSC and NSE constraints.

FSCs have been previously used to take actions in a partially observable MDP (POMDP) where the next action can depend on the agent's action-observation history (Hansen 1998). In our framework, a controller representation is learned to summarize the history for NSE prediction. As learned FSC node transitions are *Markovian*, it is easy to integrate them into MDP solution techniques such as the dual LP (Puterman 1990). FSCs also provide a more explainable model of NSEs than black-box methods.

Empirical evaluations on two domains show that our approach efficiently learns to mitigate Markovian and non-Markovian NSEs from a limited amount of historical data, outperforming the previous best method (Saisubramanian, Kamar, and Zilberstein 2020, 2022).

Problem Formulation

Consider an agent operating based on a MDP, defined by the tuple $\langle S, A, \mathrm{T}, \mathrm{r}, \gamma, b_0 \rangle$. Being in state $s \in S$ and taking action $a \in A$ causes the agent to stochastically transition to a new state s' with probability $\Pr(s'|s,a) = \mathrm{T}(s,a,s')$, and receive reward $\mathrm{r}(s,a)$. We assume an infinite-horizon setting with discount factor $\gamma < 1$ and start state distribution $b_0(s)$. The agent's policy is represented by $\pi(a|s) = \Pr(a|s)$.

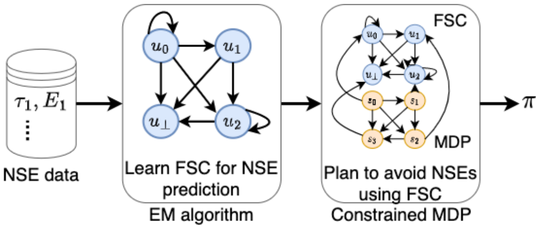


Figure 2: Overview of CASP framework.

We assume the planning setting with known transition and reward functions. The *primary objective* of the agent is to find a policy π that maximizes $V(\pi) = \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t r_t | \pi\big]$, with the optimal value denoted by V^*. We assume that the state representation has all the necessary features required to complete the primary task. A solution obtained by optimizing the primary task alone may result in NSEs.

Negative side effects (NSEs) We define NSEs similar to Saisubramanian, Zilberstein, and Kamar (2022). We consider episodic tasks where each complete trajectory $\tau = \langle s_0, a_1, s_1, \ldots, a_H, s_H \rangle$ can be of arbitrary, but finite length, terminating in some state s_H.

Definition 1. *Let* Λ *denote a* partition *of the space of all possible complete trajectories for the given MDP into mutually exclusive sets of* categories *of NSEs:* $\Lambda = \lambda_1 \cup \lambda_2 \cup \ldots \cup \lambda_K$.

Intuitively, each λ_i defines a NSE category. We assume that one set λ_j represents the absence of NSE in corresponding trajectories. Without loss of generality, we assume that a trajectory is associated with a *single NSE category*, which is generally the most severe form of NSE that occurs in the trajectory. When a trajectory $\tau \in \lambda_i$ is encountered during plan execution, we say the NSE i has been observed. We consider NSEs that are *non-Markovian* as the category may depend on the entire trajectory, in contrast to Markovian NSEs that depend on a single state-action pair (Saisubramanian, Kamar, and Zilberstein 2022). Non-Markovian NSEs are significantly richer than Markovian NSEs, and can model complex NSEs without expanding the state representation.

Practically, it is infeasible to define such partitions accurately. For example, in Figure 1, a λ_i may correspond to driving fast through puddles at least k times. This can happen in multiple ways; listing all such trajectories is infeasible. One can only observe some representative samples for different partitions and learn to generalize from observed data.

Objective We aim to mitigate NSEs by allowing for some loss (also called *slack*) in the agent's primary objective. Let $\mathcal{E} = \{\lambda_1, \ldots, \lambda_K\}$ denote the set of all trajectory partitions, $E \in \{1, \ldots, K\}$ denote the corresponding NSE category and E_t denote the category at time t. The optimization problem is noted below.

$$\max_{\pi} \sum_{s} b_0(s) V^{\pi}(s) \tag{1}$$

$$V^{\pi}(s) = \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t r_t | s, \pi\big] \tag{2}$$

$$\sum_{s} b_0(s) [V^*(s) - V^{\pi}(s)] \leq \zeta \tag{3}$$

$$\sum_{t=0}^{\infty} \gamma^t \Pr(E_t = c; \pi) \leq \alpha_c \;\; \forall c = \{1, \ldots, |\mathcal{E}|\} \tag{4}$$

ζ denotes the allowed slack on the agent's primary objective (V^*), obtained by ignoring NSEs, and Eqn. (4) constrains the expected frequency of NSE occurrence. The threshold α_c denotes the tolerance for NSE category c. The parameters ζ and α_c are typically specified by the user based on their tolerance and are used to balance the trade-off between optimizing the primary objective and avoiding NSEs.

The above problem is challenging because the constraints in Equation (4) are non-linear and non-convex in policy parameters π. In addition, it is impractical to enumerate all trajectories τ that define an NSE category. Therefore, the agent learns to estimate the probability of different categories of NSEs from historical data, as described below.

Controller Design and Learning for NSE

When the agent has no prior knowledge about the side effects of its actions, it must learn a predictive model of NSEs.

Learning about NSEs We assume that the agent has access to a dataset that contains trajectories for different NSE categories $i \in E$. Let $\hat{E}_i$ denote the collection of trajectories for NSE category i. In general, $\hat{E}_i \subseteq \lambda_i$, that is, the available data does not exhaustively list all the trajectories constituting the category i. Such data can be collected using various forms of feedback, such as by exploring the environment, from human feedback, or the past deployment of the system (Saisubramanian, Zilberstein, and Kamar 2022). For our empirical results, we used an ϵ-greedy policy using the optimal primary value function $V^\star$ to collect this data.

Our goal is to learn a classifier f, using the available dataset, that takes any trajectory τ as input and predicts the NSE category for τ. We assume *no-NSE* is also a category. Note that trajectories τ can be of variable length. We focus on using *FSCs as the classifier* representation for this multi-class classification problem.

NSE representation using FSC The existing approaches use a tabular representation for NSEs. However, this approach suffers from two key limitations: (1) it is not scalable to large problems with Markovian NSE; and (2) it cannot represent non-Markovian NSEs. To overcome these drawbacks, we propose using an FSC to learn about and compactly represent both Markovian and non-Markovian NSEs.

An FSC can compactly summarize information in a state-action trajectory and can be easily integrated into solution methods for MDPs by defining a joint-state space over the environment states and FSC nodes, also called the *cross-product MDP* (Meuleau et al. 1999a). This decoupled representation eliminates the need for updating the MDP to represent NSEs, which may require extensive testing to ensure no new risks are introduced. Furthermore, FSCs provide an explainable form for learning NSEs; empirically, a small dataset was sufficient to learn their structure.

Let $[L]$ denote the set $\{1, \ldots, L\}$ for any positive integer L. We assume there are K NSE categories, $|E| = K$.

Definition 2. *An NSE controller for a given MDP is denoted by* $\mathcal{C} = \langle \Sigma, E, U, u_{\mathfrak{s}}, u_\perp, \delta, \omega \rangle$*:*

- Σ *is a finite set of propositions representing high-level features of the problem;*

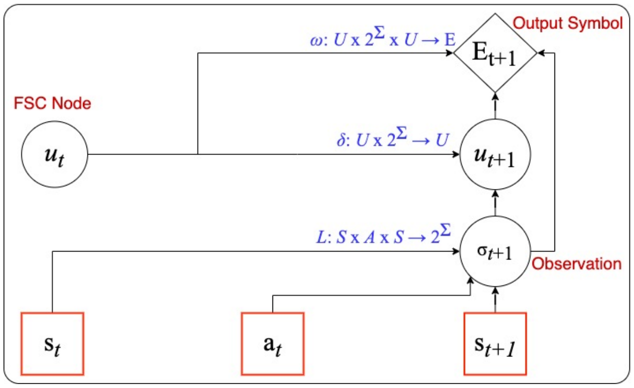

Output Symbol
ω: U x 2^Σ x $U \rightarrow$ E
E_{t+1}
FSC Node
u_t
δ: U x $2^\Sigma \rightarrow U$
u_{t+1}
L: S x A x $S \rightarrow 2^\Sigma$
σ_{t+1}
Observation
s_t
a_t
s_{t+1}

Figure 3: 2-Slice dynamic Bayesian net representing the FSC classifier. The state-action trajectory (bottom row) is observed; u_t, u_{t+1} denote the controller nodes at time t, $t+1$; σ_{t+1} is the high-level observation, E_{t+1} is the output symbol representing the observed NSE category for the input trajectory. The ω and δ are the parameters to learn.

- $E = [K] \cup \rho$ *is a finite set of output symbols that denote various NSE categories, with an empty output symbol at intermediate nodes to handle non-Markovian NSE;*
- U *is a finite set of nodes, with* $u_{\mathfrak{s}}$ *as the initial node and* $u_\perp$ *as the terminal node;*
- $\delta : U \times 2^\Sigma \rightarrow \Delta U$ *is the transition function, denoting the probability of transitioning between the nodes after receiving an observation* $\sigma \in 2^\Sigma$*, with* ΔU *denoting the distribution over successor controller nodes; and*
- $\omega : U \times 2^\Sigma \times U \rightarrow \Delta E$ *denotes the probability of an NSE category, given the nodes and input symbol.*

Figure 3 shows the relationship between different variables. The propositions, Σ, represent the high-level features of a state-action pair. The high-level observation σ corresponding to an experience (s, a, s') is determined via a labeling function $L : S \times A \times S \rightarrow 2^\Sigma$. The labeling function assigns truth values to propositions Σ, given (s, a, s').

Intuitively, the observation σ is a high-level view of low-level environment states and actions and is important for the generalizability of the learned FSC. Such labeling functions have been used for learning controllers for POMDPs (Rodrigo et al. 2019). Similar to their work, we assume a labeling function is provided as part of problem definition.

Predicting the NSE associated with an agent trajectory using an FSC involves three steps: (1) each (s, a, s') in the trajectory is mapped to an observation σ using the labeling function L; (2) the controller transitions from the current node to a successor node upon receiving σ; and (3) the controller node outputs a symbol that indicates the NSE category associated with the trajectory. For Markovian NSE, each node in the controller may be able to predict the NSE category associated with each (s, a, s') experience. For non-Markovian NSEs, all the states and actions in the trajectory must be accounted for before determining the NSE. Therefore, all intermediate nodes output ρ deterministically, and the terminal node $u_\perp$, corresponding to the end of the trajectory, will determine the NSE category.

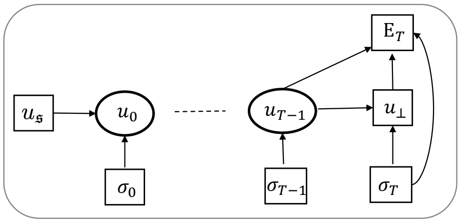


Figure 4: Markov chain representing the controller based classification of a trajectory.

Example We briefly describe the NSE prediction using FSC for the AV domain in Figure 1. Let controller nodes track the number of times the AV navigated through a puddle w/ and w/o pedestrians nearby, along with the speed and whether the goal state has been reached. The proposition set is $\Sigma = \{$puddles_pedestrians, puddles_no_pedestrians, high_speed, goal_reached$\}$. Let us first consider a *Markovian* setting where mild NSE occurs when driving fast through a puddle w/o nearby pedestrians, and severe NSE occurs when driving fast through a puddle w/ pedestrians nearby. When the AV follows the red trajectory and drives fast through the first puddle, the corresponding label is $\sigma = (T, F, T, \neg g)$, where T and F denote whether the proposition is true or false in the *current* state-action pair, which the controller uses to track the number of times the AV drove fast through puddles. The σ causes a controller transition from u_0 to $\delta(u_0, \sigma)$. The output symbol $\omega(u_0, \sigma, \delta(u_0, \sigma))$ is severe NSE. Let us now consider a *non-Markovian* version where navigating fast through puddles w/o nearby pedestrians for $> 25\%$ of its trajectory length results in a mild NSE, and driving fast through puddles with pedestrians nearby results in severe NSE. Given $\sigma = (T, F, T, \neg g)$, the output symbol $\omega(u_0, \sigma, \delta(u_0, \sigma)) = \rho$, as it is not the end of the trajectory. However, at the end of the red trajectory, $\sigma = (F, F, T, g)$, $\delta(u, \sigma) = u_\perp$, and the output $\omega(u, \sigma, u_\perp)$ is severe NSE.

Since NSEs are often discovered after deployment, defining associated FSC during design is impossible. Hence, the agent must gather information about NSE to learn the FSC.

Training data Let $\hat{E}_c$ denote the set of state-action trajectories of NSE category c. The training data for the classifier is of the form $\{(x, y)\}$ where x is the input to the classifier, and y is the true label. In our case, it is $\{(\tau, \langle u_\perp, c\rangle)\ \forall \tau \in \hat{E}_c\}$, where the trajectory τ is the input and $\langle u_\perp, c\rangle$ is the true label indicating that control must terminate in terminal node $u_\perp$ and the output symbol in the terminal node is c, denoting the NSE category associated with $\hat{E}_c$. Training data can be generated for all $\hat{E}_c$.

Learning controller parameters Given a training data point $(\tau, \langle u_\perp, c\rangle)$, the trajectory τ is converted into a sequence of high-level observations $\sigma_{0:T}$ using the labeling function L. The Markov chain connecting observed and hidden variables for the FSC is shown in Figure 4. Observed values are represented using square nodes, hidden variables using ellipses. Assume τ is a T-step trajectory. In our setting, the node at the last time step T must be terminal node $u_\perp$, and output symbol $E_T = c$. Using the principle of maximum likelihood estimation (MLE), our goal is (to avoid notation clutter, we formulate for a single training data point):

$$\max_{\delta,\omega} p(\sigma_{0:T}, u_\mathfrak{s}, u_\perp, E_T \mid \delta, \omega). \qquad (5)$$

As $u_{0:T-1}$ are hidden variables, we can treat (5) as an MLE problem with missing data and use the well-known EM algorithms to estimate parameters δ, ω (Dempster, Laird, and Rubin 1977). The EM algorithm has a particularly well suited structure for exponential family distributions (Bishop 2006) with the MLE often solvable analytically. An exponential family is a set of probability distributions with a probability density function of the form:

$$\log p(x, y \mid \theta) = D(\theta) \cdot \mathcal{T}(x, y) - C(\theta) + B(x, y),$$

where $\mathcal{T}(x, y)$ is the sufficient statistic of the data (x, y).

Assume that variables x are hidden, and y are observed (i.e., the missing data setting). Let the *expected* sufficient statistic be given as $\overline{\mathcal{T}(x, y)} = \mathbb{E}_{p(x|y;\theta)}\big[\mathcal{T}(x, y)\big]$, where θ is the current parameter estimate. The EM algorithm provides the next improved estimate $\theta^\star$ by solving the problem:

$$\theta^\star = \arg\max_{\theta' \in \Omega} \big[D(\theta') \cdot \overline{\mathcal{T}(x, y)} - C(\theta')\big]. \qquad (6)$$

Next, we show that the joint-distribution for the model in figure (4) belongs to the exponential family and characterize its sufficient statistic to formulate the equivalent optimization problem as (6). For our case, $\theta = (\delta, \omega)$. Let M denote the total nodes in our FSC, including terminal and start node. Let indices $m, n \in [M]$ be used to index over FSC nodes. Using an overcomplete representation, we define:

- u_t as M-dimensional one hot vector, with $u_t^m = 1$ means controller node is m at time t.
- Let $i \in [2^{|\Sigma|}]$ index over all observations. Let σ_t be a $2^{|\Sigma|}$-dimensional one hot vector; $\sigma_t^i = 1$ indicates observation i is true at time t.
- Let $c \in [|E|]$ index over output symbols. Let E_T be $|E|$-dimensional one hot vector defined analogously to σ_t.

FSC parameters The parameters to learn are δ, ω; $\delta(n|m, i)$ denote the probability of transitioning to node n given current node is m, and observation received is i. For non-Markovian NSE, the output symbol is only received when the current node is $u_\perp$. Let $\omega(c|m, i, u_\perp)$ denote the probability of receiving output symbol c given last node was m, current observation is i and current node is $u_\perp$. We use shorthand $\omega(c|m, i)$ by omitting $u_\perp$.

Theorem 1. *Let $\boldsymbol{u} = (u_\mathfrak{s}, u_{0:T-1}, u_\perp)$, $\boldsymbol{\sigma} = \sigma_{0:T}$. The complete data distribution $p(\boldsymbol{u}, \boldsymbol{\sigma}, E_T; \delta, \omega)$ for model in Figure 4 belongs to the exponential family, specified using:*

- $D(\delta, \omega) \leftarrow \log[\delta(n|m, i)], \log[\omega(c|m, i)]\ \forall n, m, i, c$
- *Sufficient statistic vector is:*

$$T(\boldsymbol{u}, \boldsymbol{\sigma}, E_T) \leftarrow [u_0^m \sigma_0^i]\ \forall m, i;,\ [u_{T-1}^m \sigma_T^i]\ \forall m, i;$$

$$\Big[\sum_{t=0}^{T-2} u_t^m u_{t+1}^n \sigma_{t+1}^i\Big]\ \forall m, n, i;\quad [E_T^c u_{T-1}^m \sigma_T^i]\ \forall c, m, i$$

- $B(\boldsymbol{u}, \boldsymbol{\sigma}, E_T) \leftarrow \sum_{t=0}^{T} \log p(\sigma_t); \qquad C(\delta, \omega) \leftarrow 0$

Using the above terms, the equivalent optimization problem to (6) for FSC learning is given in table 1. To avoid clutter, we show terms only for a single training data point; fi-

$$\max_{\delta,\omega}\sum_{m,i}\mathbb{E}\big[u_0^m\sigma_0^i\big]\log[\delta(m|u_{\mathfrak{s}},i)]+\sum_{m,n,i}\mathbb{E}\big[\sum_{t=0}^{T-2}u_t^m u_{t+1}^n\sigma_{t+1}^i\big]$$
$$\times\log[\delta(n|m,i)]+\sum_{m,i}\mathbb{E}\big[u_{T-1}^m\sigma_T^i\big]\log[\delta(u_\perp|m,i)]$$
$$+\sum_{c,m,i}\mathbb{E}\big[E_T^c u_{T-1}^m\sigma_T^i\big]\log[\omega(c|m,i)] \quad (7)$$
$$\sum_{n\in[M]}\delta(n|m,i)=1 \quad \forall m\in[M], i\in[2^{|\Sigma|}] \quad (8)$$
$$\sum_{c\in[|E|]}\omega(c|m,i,u_\perp)=1 \quad \forall m\in[M], i\in[2^{|\Sigma|}] \quad (9)$$
$$\delta(u_\perp|u_\perp,i)=1\ \forall i\in[2^{|\Sigma|}] \quad (10)$$
$$\delta(u_{\mathfrak{s}}|m,i)=0\ \forall m\in[M],\forall i\in[2^{|\Sigma|}] \quad (11)$$
$$\omega(\rho|u_\perp,i,u_\perp)=1\ \forall i\in[2^{|\Sigma|}] \quad (12)$$

Table 1: Optimization problem for FSC parameter learning

nal optimization problem involves summation of analogous terms for all the training data points.

The constraints (8) and (9) are standard probability normalization constraints. Constraint (10) ensures that $u_\perp$ is an absorbing node without any outgoing transitions. Constraint (11) ensures that there is no incoming transition to the starting node $u_{\mathfrak{s}}$. Constraint (12), along with constraint (10), ensure that we only receive a valid output symbol $c \neq \rho$ when the control reaches the terminal node $u_\perp$ for the first time; when the last node is $u_\perp$ and the current node is also $u_\perp$, we receive a null output symbol (ρ). We also note that although total observations are $2^{|\Sigma|}$, often many observations are infeasible in a domain. Therefore, the complexity of above program is often much lower than exponential in the number of propositions.

We show in the supplement how to compute the expected sufficient statistics using a forward-backward algorithm, similar to the well-known Baum-Welch algorithm (Bishop 2006) adapted to our setting, and use the KKT conditions (Bertsekas 1999) to solve the problem in table 1 analytically to obtain improved estimates of δ, ω parameters.

NSE Mitigation Using Dual LP for MDPs

We now show how the learned FSC can be integrated into the dual LP formulation (Altman 2021). The optimization formulation we develop approximates the problem (Eqn. (1)) as the learned FSC may not be fully accurate. We first define the cross-product MDP over the joint space $U \times S$ (Meuleau et al. 1999b). We also develop additional constraints to take into account NSE limits in (4). The transition function over the cross product MDP's state space is:

$$P(u', s'|u, s, a) = \mathrm{T}(s, a, s')\delta(u'|u, \sigma = L(s, a, s')),$$

where $L(\cdot)$ is the labeling function. The reward function is unaffected by the controller state and remains the same as $r(s, a)$. The probability of NSE $E_t = c$ is:

$$P(E_t = c|u_{t-1}, u_t = u_\perp, s_{t-1}, a_{t-1}, s_t) =$$
$$\omega\big(c|u_{t-1}, u_\perp, \sigma_t = L(s_{t-1}, a_{t-1}, s_t)\big). \quad (13)$$

$$\max_{\{y(\cdot)\}}\sum_{u,s,a} r(s,a)y(u,s,a) \quad (14)$$

//Dual LP flow constraints

$$\sum_a y(u',s',a)=b_0(u',s')$$
$$+\gamma\sum_{u,s,a}P(u',s'|u,s,a)y(u,s,a)\forall(u',s') \quad (15)$$
$$y(u,s,a)\geq 0 \quad \forall(u,s,a) \quad (16)$$

//NSE frequency computation

$$y(c)=\sum_{u,s,a}y(u,s,a)\sum_{s'}P(u'=u_\perp,s'|u,s,a)\times$$
$$\omega(c|u,u_\perp,\sigma=L(s,a,s'))\ \ \forall c\in E \quad (17)$$

//NSE satisfaction constraints

$$y(c)\leq\alpha_c\ \ \forall c\in E \quad (18)$$

//Primary objective slack

$$\sum_s b_0(s)V^*(s)-\sum_{u,s,a}r(s,a)y(u,s,a)\leq\zeta \quad (19)$$

Table 2: Dual Linear Program for Safe Policy Optimization

The dual LP for the cross-product MDP incorporating NSE constraints is given in Table 2. The structure and interpretation of this dual LP is similar to the standard dual LP for MDPs (Puterman 1990), with the occupancy measures defined over the cross-product state space $U \times S$. The occupancy measures $y(u, s, a)$ denote the total expected number of times controller state is u, MDP state is s, and action taken is a (represented by 'dual LP flow constraints'). $b_0(u, s)$ denotes the probability of starting in controller state u and MDP state s. We assume the agent observes the joint state (u, s). The policy π can be extracted from the optimal solution $y^\star$ as follows: $\pi^\star(a|u, s) = {}^{y^\star(u,s,a)}/_{\sum_{a'} y^\star(u,s,a')}$. Constraints (17)-(19) are the major differences from the standard dual LP. Constraint (17) compute the probability of different NSEs $c \in E$ as per our learned FSC. Following results show the correctness of constraints (17), (18).

Definition 3. *Let $y(c; \pi)$ be the total expected number of times NSE c is encountered as per policy π:*

$$y(c;\pi)=\sum_{t=0}^{\infty}\gamma^t P(E_{t+1}=c;\pi) \quad (20)$$

In the above, we assume that it takes at least one time step to emit an output symbol, as it takes at least one step to reach the terminal node $u_\perp$ from the start node $u_{\mathfrak{s}}$.

Proposition 1. *The occupancy measure $y(c; \pi)$ for an NSE $c \in E$ as per the policy π can be computed as:*

$$y(c;\pi)=\sum_{u,s,a}y(u,s,a;\pi)\sum_{s'}P(u'=u_\perp,s'|u,s,a)\times$$
$$\omega(c|u,u_\perp,\sigma=L(s,a,s')) \quad (21)$$

Proof in Appendix. As a result, constraints (17), (18) model the constraint (4) in our original problem, and constraint (19) models the constraint (3). Thus program in Table 2 approximately solves the problem (1) (up to accuracy afforded by the learned FSC). We empirically run simulations to estimate $y^{\star}(c)$ to test if the final policy avoids NSEs.

Empirical Evaluation

We evaluate the effectiveness of our approach, controller-assisted safe planning (CASP), in learning to predict and mitigate Markovian and non-Markovian NSEs. We assume Markov state representation for the primary objective. In the interest of clarity, we test with three NSE categories: *mild NSE, severe NSE,* and *no NSE*. Each action/trajectory can result in a mild, severe, or no NSE.

Baselines We compare the performance of our approach with three baselines: (1) executing the *Initial* policy that optimizes the primary objective, with no NSE learning involved; (2) a multi-objective approach to mitigate NSEs (*LMDP*) (Saisubramanian, Kamar, and Zilberstein 2020) with a perfect model of NSE (*LMDP Optimal*); and (3) LMDP with a predictive model of NSE learned using approval feedback (*LMDP Learned*). Since the LMDP can handle only Markov NSEs, we calculate the non-Markovian NSEs encountered by simulating the policy for comparison.

In our experiments, we optimize costs, which are negations of the reward. We solve the planning problem using `Advanced Process Optimizer` (APOPT) solver, with the controller learned using EM algorithm and $\gamma = 0.99$. All experiments were conducted on an Ubuntu machine with 80GB RAM. Following the planning, we compute average NSE values by performing 10K simulations (e.g., *average NSE* = 0.5 implies 50% of 10K simulations encountered NSE). Our code is publicly available[1].

Boxpushing In this domain, the agent aims to push a box as quickly as possible to a goal location (Saisubramanian, Kamar, and Zilberstein 2020). The state is represented by $\langle x, y, b_x, b_y, b, c\rangle$, with x, y denoting the agent's position, b_x, b_y denoting the box position, b is a Boolean variable indicating whether the agent is pushing the box, c indicates the type of surface: rug or plain. The agent can move in all four directions, each costing $+1$. The agent can also load the box with 'pick up box' action that costs $+2$, and wrap the box with a protective sheet using 'wrap box' action that costs $+5$. 'Pick up' and 'wrap box' actions are deterministic. The 'move' actions succeed with probability 0.9 and slide to a neighboring cell with probability 0.1. *Markovian NSE* occurs when the agent pushes the box over the rug. In *Non-Markovian NSE*, the effects are mild when $1-25\%$ rug area is dirtied, and severe if $> 25\%$ is dirtied when the agent completes its task. We experiment with grid size 15×15.

Navigation Our second domain is the AV navigation described in Figure 1, where the AV aims to navigate quickly to a goal location (Saisubramanian, Kamar, and Zilberstein 2020). The AV can move in all four directions and navigate at two speeds: slow and fast. Driving slow costs $+2$, driving fast costs $+1$. Each state is represented by $\langle x, y, \text{speed, pedestrian, puddle}\rangle$. Pedestrian and puddle are Boolean variables. The AV's move actions succeed with probability 0.9 or fail with probability 0.1 and slide to a neighboring cell. The agent can transition between any speed deterministically. *Markovian NSE* occurs when driving fast through puddles. *Non-Markovian NSE*: mild NSE occurs when the AV drives fast through puddles, without pedestrians in the vicinity, for $> 25\%$ of its route. Driving fast through puddles with pedestrians nearby results in severe NSE. We experiment with grid size 15×15.

Domain	#Nodes	F-1 scores			Accuracy
		No NSE	Mild	Severe	(%)
Boxpushing (15×15)	4	0.65	0.48	0.45	67.00
	5	0.67	0.49	0.52	68.00
	6	0.68	0.52	0.54	69.00
	7	0.80	0.75	0.76	86.00
	8	0.86	0.84	0.83	91.40
	9	0.89	0.89	0.89	91.30
Navigation (15×15)	4	0.51	0.51	0.67	67.03
	5	0.47	0.48	0.65	66.69
	6	0.52	0.67	0.70	74.32
	7	0.85	0.85	0.87	89.28
	8	0.90	0.90	0.91	92.78
	9	0.87	0.86	0.88	91.70

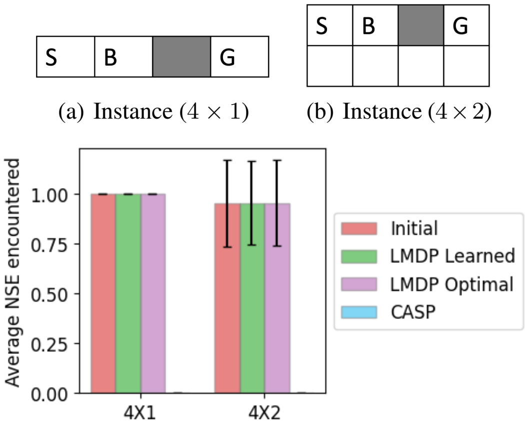
Table 3: F-1 scores for each NSE category and overall accuracy with varying controller sizes (# nodes) on two domains.

Results and Discussion

Learning FSC We evaluate the effectiveness of our approach in learning a FSC to predict NSEs using F-1 scores for each NSE category and overall prediction accuracy, as we vary the controller size (Table 3). We use 75 trajectories for training and 305 for testing in the boxpushing domain, and 300 for training and 1155 for testing the navigation domain. We use more trajectories for navigation domain, since the trajectories are relatively longer and the NSE condition is more complex. While the accuracy may improve as we increase the controller size, it also increases the training time (Fig. 6 in Appendix). Hence, we choose the smallest size that achieves comparable performance across NSE categories and $\sim 90\%$ accuracy. Based on these results, we use a controller size of eight nodes for the boxpushing domain and seven nodes for the navigation domain.

Effectiveness in slack utilization Consider two simple boxpushing instances (4×1 and 4×2), where the shaded area denotes the rug, B denotes box location, S and G denote start and goal respectively (Figure 5). We evaluate with Markovian NSE, slack $\zeta = 5$, and NSE threshold $\alpha = 0$. The *Initial* policy always produces NSE. To avoid the NSE, the agent can wrap the box, incurring an additional cost $+5$, which matches the allowed slack. While our approach with a learned FSC avoids NSE by wrapping the box, *LMDP* cannot avoid NSE even with a perfect NSE model (LMDP Optimal). This is because of the fundamental difference in how the two approaches distribute the slack. The agent can only execute actions within the allowed slack in each state. Our approach allows the slack to be used in whole in any state, so the agent can use the wrap action to avoid the NSE. However, *LMDP* distributes the global slack to each state using $\eta = (1-\gamma)\zeta$, where η is the slack for each state, which can lead to harsh pruning of the policy space and result in poor performance (Pineda, Wray, and Zilberstein 2015). In our

[1] https://github.com/SriAish/NSE_PlanningAgent



(a) Instance (4 × 1) (b) Instance (4 × 2)

(c) Average Markovian NSE with standard deviation.

Figure 5: Simple boxpushing instances, with $\zeta = 5$, demonstrate limitation of LMDP approach in mitigating Markovian NSE, due to its slack distribution; CASP has zero NSE

Domain	Approach	Slack	Average NSE
Boxpushing	Initial Policy	-	0.9700 ± 0.1626
	LMDP Learned	15%	0.9729 ± 0.1623
		20%	0.9720 ± 0.1649
		25%	0.9708 ± 0.1683
	LMDP Optimal	15%	0.9735 ± 0.1606
		20%	0.9715 ± 0.1663
		25%	0.9688 ± 0.1738
	CASP (#Nodes: 4)	15%	–
		20%	0
		25%	0
Navigation	Initial Policy	-	1.00 ± 0
	LMDP Learned	15%	1.0000 ± 0
		20%	1.0000 ± 0
		25%	1.0000 ± 0
	LMDP Optimal	15%	1.0000 ± 0
		20%	1.0000 ± 0
		25%	1.0000 ± 0
	CASP (#Nodes: 4)	15%	0.0005 ± 0.0223
		20%	0.0005 ± 0.0223
		25%	0.0005 ± 0.0223

Table 4: Effect of varying slack on *Markovian NSE*, when NSE threshold $\alpha = 0$.

setting, the agent cannot avoid NSE as the wrap action violates the slack allotted to any state. This slack distribution method is a fundamental limitation in *LMDP* that affects its performance. This experiment demonstrates effective slack utilization by our approach to mitigate NSEs, when feasible.

Mitigating Markovian NSE All Markovian NSEs are assumed to have same severity. Results in Figure 5 and Table 4 show average Markovian NSE, along with standard deviation, for $\alpha = 0$ when slack is varied as % of primary objective value. Since Markovian NSEs are relatively easier to predict, they can be avoided with a smaller controller size.

Mitigating non-Markovian NSE Tables 5 and 6 show the results on non-Markovian NSEs for both the domains, with NSE threshold $\alpha = 0$ for both mild and severe NSE, and varying slack. Controller sizes were selected based on Table 3. For the boxpushing domain, CASP approach did not find a solution with 15% slack but could avoid NSE when the slack increased. *LMDP* could not avoid the NSE, despite increasing the slack. This shows that besides its limitation in effectively using the slack, *LMDP* cannot mitigate non-Markovian NSEs. We also test the effect of varying NSE thresholds and controller sizes on the performance (Fig. 7 in Appendix). Results with seven nodes for navigation and eight nodes for the boxpushing domain avoid all NSEs, for the three (mild, severe) threshold configurations we tested.

Approach	Slack	Average NSE	
		Mild	Severe
Initial Policy	-	0.02 ± 0.16	0.95 ± 0.23
LMDP Learned	15%	0.02 ± 0.16	0.95 ± 0.23
	20%	0.03 ± 0.16	0.95 ± 0.23
	25%	0.03 ± 0.16	0.94 ± 0.23
LMDP Optimal	15%	0.95 ± 0.22	0.02 ± 0.16
	20%	0.03 ± 0.16	0.95 ± 0.22
	25%	0.03 ± 0.16	0.95 ± 0.23
CASP (#Nodes: 8)	15%	-	-
	20%	0	0
	25%	0	0

Table 5: Effect of varying slack on non-Markovian NSE in boxpushing domain with $\alpha = 0$ for mild and severe NSE.

Approach	Slack	Average NSE	
		Mild	Severe
Initial Policy	-	0	1
LMDP Learned	15%	0	0.99 ± 0.017
	20%	0	0.99 ± 0.014
	25%	0	0.94 ± 0.241
LMDP Optimal	15%	0	0.99 ± 0.009
	20%	0	0.99 ± 0.009
	25%	0	0.99 ± 0.014
CASP (#Nodes: 7)	15%	0	0
	20%	0	0
	25%	0	0

Table 6: Effect of varying slack on non-Markovian NSE in navigation domain with $\alpha = 0$ for mild and severe NSE.

Summary and Future Work

We present CASP, a paradigm to learn and mitigate Markovian and non-Markovian NSEs, with bounded-performance guarantees for the primary objective value and NSE occurrence. We use an FSC to learn about, compactly represent, and predict NSEs, using EM algorithm. The problem of mitigating NSEs is formulated as a constrained MDP, and an NSE-minimizing policy is computed by integrating the FSC with our planning model. Our results with varying slack, controller sizes, NSE thresholds on Markovian and non-Markovian NSEs demonstrate the benefits of our approach. In the future, we aim to extend our technique to partially observable settings and to handle noise in the training data. Extending our approach to handle continuous state space is another interesting direction for future research.

Acknowledgments

This research/project is supported by the National Research Foundation, Singapore and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-RP-2020-017), US National Science Foundation USDA-NIFA grant number 2021-67021-35344, and Kohli Center on Intelligent Systems, IIIT Hyderabad.

References

Alizadeh Alamdari, P.; Klassen, T. Q.; Toro Icarte, R.; and McIlraith, S. A. 2022. Be Considerate: Avoiding Negative Side Effects in Reinforcement Learning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 18–26.

Altman, E. 2021. *Constrained Markov Decision Processes*. ISBN 9781351458245.

Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; and Mané, D. 2016. Concrete problems in AI safety. *CoRR*, abs/1606.06565.

Bertsekas, D. 1999. *Nonlinear Programming*. Athena Scientific.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag. ISBN 0387310738.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1): 1–38.

Dietterich, T. G. 2017. Steps toward robust artificial intelligence. *AI Magazine*, 38(3): 3–24.

Hansen, E. A. 1998. Solving POMDPs by Searching in Policy Space. In *International Conference on Uncertainty in Artificial Intelligence*, 211–219.

Krakovna, V.; Orseau, L.; Martic, M.; and Legg, S. 2019. Penalizing Side Effects using Stepwise Relative Reachability. In *AI Safety Workshop, IJCAI*.

Krakovna, V.; Orseau, L.; Ngo, R.; Martic, M.; and Legg, S. 2020. Avoiding Side Effects By Considering Future Tasks. In *Advances in Neural Information Processing Systems*.

Meuleau, N.; Kim, K.; Kaelbling, L. P.; and Cassandra, A. R. 1999a. Solving POMDPs by Searching the Space of Finite Policies. In Laskey, K. B.; and Prade, H., eds., *International Conference on Uncertainty Artificial Intelligence*, 417–426.

Meuleau, N.; Peshkin, L.; Kim, K.-E.; and Kaelbling, L. P. 1999b. Learning Finite-State Controllers for Partially Observable Environments. In *International Conference on Uncertainty in Artificial Intelligence*, 427–436.

Pineda, L. E.; Wray, K. H.; and Zilberstein, S. 2015. Revisiting Multi-Objective MDPs with Relaxed Lexicographic Preferences. In *Proceedings of the AAAI Fall Symposium Series*.

Puterman, M. L. 1990. Markov Decision Processes. *Handbooks in operations research and management science*, 2: 331–434.

Rodrigo, T. I.; Waldie, E.; Klassen, T.; Valenzano, R.; Castro, M.; and McIlraith, S. 2019. Learning reward machines for partially observable reinforcement learning. In *Advances in Neural Information Processing Systems*.

Saisubramanian, S.; Kamar, E.; and Zilberstein, S. 2020. A Multi-Objective Approach to Mitigate Negative Side Effects. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, 354–361.

Saisubramanian, S.; Kamar, E.; and Zilberstein, S. 2022. Avoiding Negative Side Effects of Autonomous Systems in the Open World. *Journal of Artificial Intelligence Research*, 74: 143–177.

Saisubramanian, S.; Roberts, S. C.; and Zilberstein, S. 2021. Understanding User Attitudes Towards Negative Side Effects of AI Systems. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, 368:1–368:6.

Saisubramanian, S.; Zilberstein, S.; and Kamar, E. 2022. Avoiding negative side effects due to incomplete knowledge of AI systems. *AI Magazine*, 42(4): 62–71.

Zhang, S.; Durfee, E. H.; and Singh, S. 2020. Querying to Find a Safe Policy Under Uncertain Safety Constraints in Markov Decision Processes. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, 2552–2559.

Zhang, S.; Durfee, E. H.; and Singh, S. P. 2018. Minimax-Regret on Side Effects for Safe Optimality in Factored Markov Decision Processes. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 4867–4873.