Mitigating Negative Side Effects

Thesis submitted in partial fulfillment
of the requirements for the degree of

*Master of Science in **Computer Science and Engineering** by Research*

by

Aishwarya Srivastava
20171046
aishwarya.srivastava@research.iiit.ac.in

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad - 500 032, INDIA
May, 2023

Copyright © Aishwarya Srivastava, 2023
All Rights Reserved

International Institute of Information Technology

Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "**Mitigating Negative Side Effects**" by **Aishwarya Srivastava**, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Dr. Praveen Paruchuri

To my Mom, Dad, and Brother,

who have supported me throughout my education.

Thanks for always motivating me to do my best.

Acknowledgments

I am grateful to my advisor, Dr. Praveen Paruchuri, for his guidance and support throughout my research journey. I would especially like to thank him for his advice while finding a research problem statement and for the motivation to work through the various ups and downs I faced during this journey. I have learned a lot from him, including how to analyze a problem statement's feasibility and asses its impact. He also taught me how to evaluate my solutions and make them better.

I would also like to sincerely thank Dr. Akshat Kumar from Singapore Management University (SMU) and Dr. Sandhya Saisubramanian from Oregan State University for their constant support both during the development of this research and while working on my research paper. I am grateful to get a chance to collaborate with them. Dr. Sandhya's expert knowledge of Safe AI systems helped me expedite my understanding of the problem statement and come up with a solution. Dr. Akshat's deep expertise in mathematical concepts and various algorithms helped me a lot while building the solution. They and my advisor helped me a lot while writing the research paper and testing the solution, and they were always supportive and understanding.

I sincerely thank Anshita Khandelwal for her support while writing the research paper and thesis. I would also like to thank my friends Alok Kar, Sanjana Sunil, Samhita Kanaparthy, Mona, and Ananya Arun for their constant support and positivity. I would take this opportunity to thank all my teachers that had a role in shaping me into the person I am today. I would also like to thank all the staff and administration at IIIT Hyderabad for helping me promptly throughout the college.

Lastly, I would like to thank my parents, Rashmi Srivastava and Saumitra Priya Srivastava, and my older brother, Yash Srivastava, for supporting me through all the ups and downs of my research journey and providing me with their unconditional love and positivity. I faced many challenges throughout this journey which I have overcome with the help of my professors, friends, and family. I am glad this thesis has reached this stage of completion.

Abstract

Autonomous systems perform various tasks across different industries ranging from finance to healthcare to space applications. However, these systems are often deployed in the open world, where it is hard to obtain complete specifications of the objectives and constraints. Operating based on an incomplete model can produce undesired effects, i.e., Negative Side Effects (NSEs). Negative side effects affect the system's safety and reliability and can be of two types: Markovian and non-Markovian.

In this thesis, we try to mitigate negative side effects in environments modeled as Markov decision processes (MDPs). Unlike previous works in this area that associate negative side effects with state-action pairs, our framework associates them with entire trajectories, which is more general and captures non-Markovian dependence on states and actions. Non-Markovian negative side effects are produced when the agent executes a certain sequence of actions in the deployed environment. Prior works mitigate Markovian negative side effects and can not be easily extended to non-Markovian negative side effects.

We build a framework, Controller-Assisted Safe Planning (CASP), for mitigating **the non-Markovian** negative side effects. Our primary contributions are:

1. We design a model based on Finite State Controllers (FSCs) that can predict the severity of negative side effects for a given trajectory.

2. We learn the model parameters using observed data containing state-action trajectories and the severity of the associated negative side effects. The model is learned such that it generalizes well to unseen data. Information about negative side effects is gathered through Oracle feedback and compactly represented as a finite state controller.

3. We develop a constrained MDP model that uses information from both the underlying MDP and the learned model for planning while avoiding negative side effects.

Our empirical evaluation demonstrates the effectiveness of our approach in learning and mitigating Markovian and non-Markovian negative side effects.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We begin this chapter with an overview of the problem of mitigating negative side effects, which we address in this thesis, and the motivation behind it. We then formally define negative side effects and briefly summarize our contributions.

1.1 Motivation

Autonomous systems are widely used across different industries [4, 41, 9, 33]. Some examples of autonomous systems used include self-driving cars, cooling data centers, predicting stock prices, and giving personalized suggestions. As autonomous systems are increasingly deployed in open-world environments, obtaining a perfect description of the target environment becomes practically infeasible [8]. Consequently, these systems often operate based on an incomplete model of the real world. Rigorous system design and testing are typically followed to ensure that the model includes all necessary details relevant to the agent's primary task. However, additional details that are considered unrelated to the primary task may be missing [30, 21]. Operating based on such incomplete models may produce *Negative Side Effects* (NSEs). Negative side effects affect the system's safety and reliability.

Addressing negative side effects is gaining increased attention since it affects the safety and reliability of deployed AI systems. Prior works mitigate Markovian negative side effects through model and policy updates [26], by constraining actions [40, 39], by minimizing deviations from a baseline [13, 36], by giving incentives to the agent to preserve the ability to perform future tasks in the environment [14], and by considering the influence of actions on other agents in the environment [1]. These approaches inherently assume that the state representation is sufficiently expressive to recognize and learn about immediate negative side effects.

In the real world, however, the negative side effects may be associated with a trajectory, and the state representation may not be sufficient for the negative side effects to be Markovian, particularly when the features determining the negative side effects are unrelated to the agent's primary task. Extending the existing approaches to mitigate non-Markovian negative side effects is not straightforward since the negative side effect penalty associated with a trajectory may not be decomposable into additive penalties associated with each state-action pair [27]. It may be computationally expensive to expand the state representation for the negative side effects to be Markovian, as the expanded representation may make the primary planning task intractable.

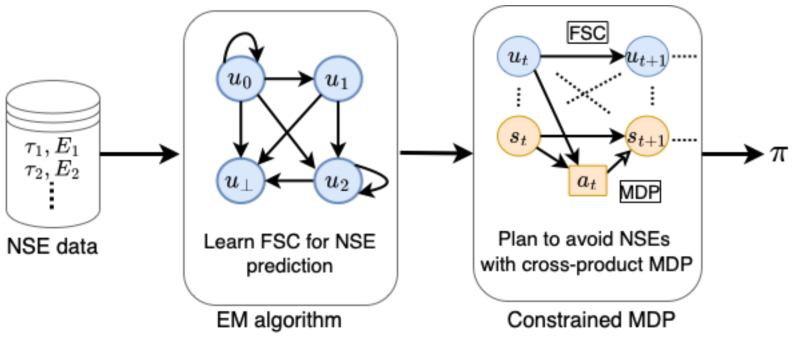


Figure 1.1: Overview of CASP framework for mitigating NSEs using FSCs.

We propose Controller-Assisted Safe Planning (CASP), a framework to mitigate the impact of *Non-Markovian negative side effects*—the state representation is assumed to be *Markovian* for the agent's primary task, but the negative side effects may be non-Markovian with respect to this representation. The problem is formulated as a *constrained Markov decision process* (CMDP), with constraints on negative side effect occurrences and deviation from the initial objective value, denoted by the slack. The slack constraint indicates the maximum allowed deviation from the optimal expected value of the objective when the agent updates its policy to mitigate negative side effects. Since the agent has no prior knowledge about negative side effects, it must learn a predictive model of negative side effects, which is then used to avoid them.

Specifically, our approach uses a three-step method to detect and mitigate negative side effects (Figure 1.1):

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) negative side effects using a finite state controller (FSC) and the EM algorithm [7];

3. The agent re-plans to mitigate negative side effects using the learned model, given a tolerance threshold and a slack. The agent solves a constrained MDP using a variant of the standard dual linear program (dual LP) for MDPs [19] to factor in the learned FSC and negative side effect constraints.

Finite state controllers 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 [11]. Our framework uses a controller representation to summarize the history for the negative side effect prediction task. As learned controller node transitions are *Markovian*, it is easy to integrate them into MDP solution techniques such as the dual LP [19]. Finite state controllers also provide a more explainable negative side effect model than black-box methods using deep neural networks.

Empirical evaluations on two domains (Boxpushing and Navigation) show that our approach efficiently learns to mitigate Markovian and non-Markovian negative side effects from a limited amount of historical data, outperforming the previous best method [26, 27]. Our approach can also be used to mitigate Markovian negative side effects with no changes to the formulation. In fact, our results show that our approach is as effective as or better than a recent approach that targets Markovian negative side effects specifically.

1.2 Negative Side Effects

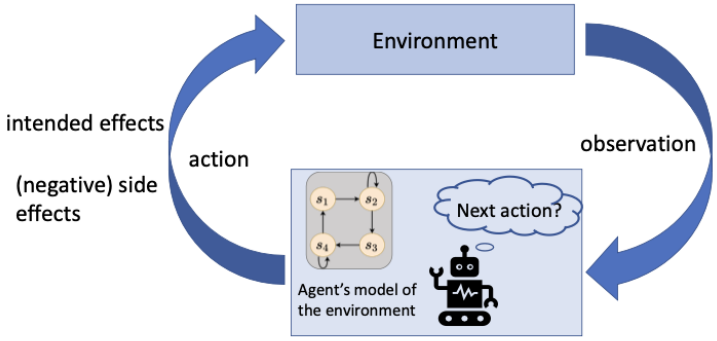


Figure 1.2: Negative side effects of an agent's behavior. [30]

Negative side effects are the unintended, undesired consequences of an agent's action or sequence of actions due to an incomplete model of the real world where it operates. They are often discovered after deployment [3, 1, 13, 28, 12]. Model incompleteness may arise in the form of under-specified objectives or missing constraints due to the limited availability of information or due to unintentional overlooking of details.

Definition 1. *Negative side effects are undesired effects of an agent's actions that occur in addition to the agent's intended effects when operating in the open world. [30] (Figure 1.2).*

The negative side effects may be Markovian or non-Markovian, depending on the problem setting [27]. Markovian negative side effects are those associated with the immediate execution of an action in a state. Non-Markovian negative side effects are associated with a sequence of actions. Many real-world domains are characterized by non-Markovian negative side effects.

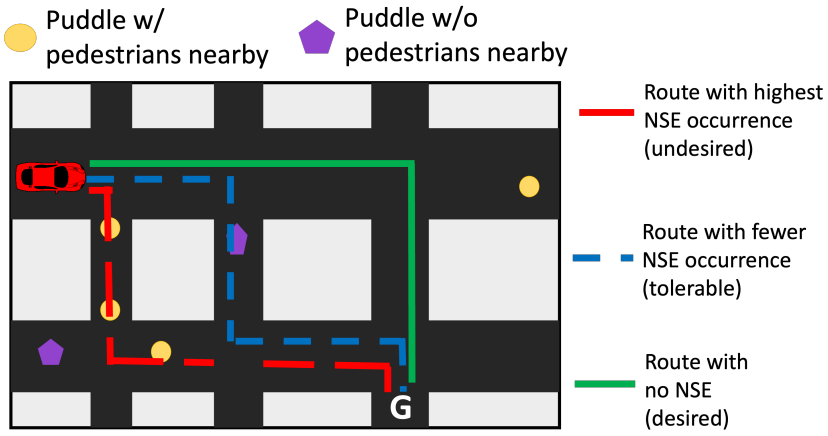


Figure 1.3: Illustration of a non-Markovian negative side effect in a driving domain, in which the negative side effect is associated with the frequency of driving fast through puddles along the route.

For example, consider an autonomous vehicle (AV) that aims to quickly navigate to a goal location while complying with the traffic rules (Figure 1.3). While the autonomous vehicle'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. When operating based on this model, the autonomous vehicle may drive fast through puddles, producing a negative side effect. While the user may be willing to tolerate this behavior occasionally, they may not be willing to tolerate the autonomous vehicle frequently splashing water on nearby pedestrians as

a side effect of driving fast through puddles. Besides affecting the user's travel experience, this behavior may also damage the vehicle.

Since the model has no information about the negative side effects, the autonomous vehicle's state representation may not include a feature indicating the number of times it has driven fast through puddles so far. Thus the severity of a negative side effect occurrence, in this case, depends on the agent's trajectory and is, therefore, *non-Markovian* with respect to its state representation, which is Markovian for the navigation task.

1.3 Contributions

Our primary contributions are:

1. We design a model based on Finite State Controllers (FSCs) that can predict the severity of negative side effects for a given trajectory.

2. We learn the model parameters using observed data containing state-action trajectories and the severity of the associated negative side effects. The model is learned such that it generalizes well to unseen data. Information about negative side effects is gathered through Oracle feedback and compactly represented as a finite state controller.

3. We develop a constrained MDP model that uses information from the underlying MDP and the learned model for planning while avoiding negative side effects.

4. We evaluate the performance of our approach on two domains, Boxpushing and Navigation.

1.4 Thesis Organization

Chapter 2 will investigate and discuss the characteristics of negative side effects, the challenges in avoiding them, and various existing algorithms in the literature for mitigating negative side effects. Our contributing chapters are 3, 4, 5 and 6. Chapter 3 introduces the optimization problem for mitigating the negative side effects. Chapter 4 and 5 discuss our novel algorithm for learning finite state controllers to represent negative side effects. Chapter 6 discusses how finite state controllers can be integrated with an MDP to approximately solve the optimization problem in Chapter 3.

Chapter 2

Related Work

We begin this chapter by discussing the characteristics of negative side effects, which are essential to understand better the negative side effects we are mitigating. We will then discuss the challenges in mitigating the negative side effects. Furthermore, we briefly introduce some existing approaches to mitigate negative side effects.

2.1 Taxonomy of Negative Side Effects

Negative side effects are typically identified after deployment and affect a system's safety and reliability. Learning and avoiding negative side effects is an emerging research area. [10, 12, 13, 23, 36, 26, 32, 39, 20, 15]. Numerous studies have also focused on the challenge of building safe and reliable AI systems. [3, 24, 31, 35]. Understanding the characteristics of negative side effects can help us design better frameworks for learning and mitigating them [30]. We briefly discuss the various characteristics of negative side effects:

- **Severity:** The severity of negative side effects can range from mild to safety-critical failures. Mild side effects can generally be ignored, whereas safety-critical failures require suspension of deployment. Our research concentrates on the negative side effects which lie in between them.

- **Reversibility:** Reversible negative side effects are the ones that can be undone through the agent's actions or external interventions. An example of reversible side effects is damage caused to an autonomous vehicle by driving fast through puddles which a mechanic can repair.

- **Avoidability:** Negative side effects that are impossible to avoid while achieving the primary goal are called unavoidable negative side effects. An example of unavoidable side effects is when all the roads leading to the goal location for an autonomous vehicle have puddles.
- **Frequency:** The frequency of negative side effects depends on the environment's conditions and the agent's policy. Most of the research in this area concentrates on mitigating frequent negative side effects it is easier to collect data for them.
- **Stochasticity:** The occurrence of the negative side effect itself might be deterministic or stochastic. Deterministic negative side effects always happen if certain preconditions are met. For example, if every time an autonomous vehicle drives fast through a puddle, it damages the vehicle, then the negative side effect is deterministic. However, if the damage is only caused 30% of the time, the negative side effects are stochastic.
- **Observability:** Negative side effects may be fully observable, partially observable, or unobservable. Observability of negative side effects does not mean the agent identifies them. For example, the agent might observe that the autonomous vehicle has been damaged but might not realize the damage is undesirable.
- **Exclusivity:** Negative side effects might prevent the agent from completing the primary goal, i.e., either the negative side effect occurs or the primary goal is completed. However, often negative side effects do not prevent the agent from fulfilling the primary goal.

Our research looks at frequent, deterministic, avoidable, non-interfering negative side effects that are not safety-critical but have a severe impact and cannot be ignored.

2.2 Challenges in Avoiding Negative Side Effects

Agents are either trained in a simulator or operate based on models. Practical challenges in model specification cause the agent to reason based on incomplete models of the real world. Simulations used to train agents also have differences from the real world. Agents operating based on incomplete information may not always behave as intended leading to negative side effects [15]. The model incompleteness may arise due to the following:

1. Difficulty in identifying negative side effects a priori.

2. The environment used to train the agent differs from where the agent is deployed. For example, the environment the agent is deployed in might has more obstacles than the environment agent was trained in. [8]

3. The negative side effect may be a violation of a user's preference. Learning and encoding each user's preferences before deployment is not simple. [21]

Another major challenge in avoiding negative side effects is feedback collection. The agent is unaware of the negative side effects and requires feedback to avoid them. The approach used to avoid negative side effects might not be sample efficient and may require large amounts of data. The approach may also require the data in specific formats. Thus, feedback collection is an expensive process. Moreover, the feedback might be biased or delayed affecting the approach's performance.

2.3 Avoiding Negative Side Effects

Prior works mitigate negative side effects through model and policy updates, constraining actions, minimizing deviations from a baseline, giving incentives to the agent to preserve the ability to perform future tasks in the environment [14], and considering the influence of actions on other agents in the environment [1]. This section briefly discusses some of the approaches to mitigate negative side effects.

2.3.1 Model and Policy Updates

Negative side effects depend on the agent's trajectory, which depends on the agent's policy. The agent's policy is learned based on reasoning models. Therefore, a simple way to avoid negative side effects would be to update the reasoning model so that the policy learned by the agent avoids negative side effects.

A form of an update is updating the reward function. In one such approach [10], the agent is assumed to be aware of possible reward misspecification, and the reward function is redesigned. The agent uses the designed reward as the indicator of the intended reward and learns the true reward using inverse reinforcement learning techniques. However, redesigning the reward function may degrade the agent's performance for the primary goal.

In domains where the side effects are not safety-critical, the impact on the primary goal can be minimized by adding a penalty function corresponding to negative side effects to the agent's model.

Adding a separate penalty function exploits the reliability of the existing model for the agent's primary goal. An example of such an approach is a multi-objective formulation [26]. In the formulation, the objectives have a lexicographic ordering for priority, and a higher priority is given to the primary objective. A reward function encodes information about the negative side effects. The model is updated with the learned reward function, and the agent's new policy is computed to avoid negative side effects as much as possible within an allowed slack.

2.3.2 Minimizing Deviations from a Baseline

Another class of approaches defines a penalty function for negative side effects as a measure of deviation from a baseline state. The deviation measure reflects the extent of disruption to the environment. The agent is expected to minimize the deviation while pursuing its goal, thus avoiding the negative side effects. It can be formulated as a multi-objective formulation with scalarization. We can adjust the agent's sensitivity by tuning the weights for scalarization.

Various candidates have been proposed for baseline states. Utilizing start state and inaction in a state for baseline and reachability-based metrics to measure the deviation was introduced in [13] and [32]. The performance of resulting algorithms is susceptible to the metric used to calculate deviations. Moreover, using the relative reachability approach [13] in settings more complex than grid worlds is not straightforward. Attainable utility [36] estimates the impact of negative side effects as the shifts in an agent's ability to optimize for secondary objectives. These approaches assume that the state representation is sufficient to compute the deviations.

2.3.3 Constrained Optimization

Negative side effects occur when the agent alters environmental features that were not expected or intended to be altered. This can be addressed by constraining the features that the agent can alter. Minimax-regret querying [40] considers a factored state representation and characterizes side effects as changes in the features of the environment that may negatively surprise a human observer. This approach assumes the agent's model includes uncertainty about an altering feature's desirability. A policy is computed assuming all the uncertain features are "locked" for alteration. The agent queries the human to decide which features can be altered and recomputes a policy if no policy exists. In an extension of the approach [39], the agent checks if the negative side effect is unavoidable. If it is unavoidable, the agent ceases operation.

Chapter 3

Mitigation of Negative Side Effects

In this chapter, we introduce the optimization problem to mitigate negative side effects (NSEs), which will be simplified in the further chapters to have a tractable optimization problem. The optimization problem also involves controlling the change in reward. We also introduce MDPs and the mathematical definition and categorization of negative side effects. We assume the environment is defined as a Markov decision process (MDP) for the optimization problem. All discussions in the further chapters assume that the environment is modeled as an MDP.

3.1 Markovian Decision Processes

Consider the model in Figure 3.1 with discrete time steps. The agent has an initial state and a reward value in each time step. Based on the current state(S_t), the agent takes action (A_t) from the set of allowed actions(A) and interacts with the environment. Based on the action, the environment gives a numerical signal(R_{t+1}) to the agent, and the state of the environment changes to a new state(S_{t+1}) from the set of states(S) of the MDP.

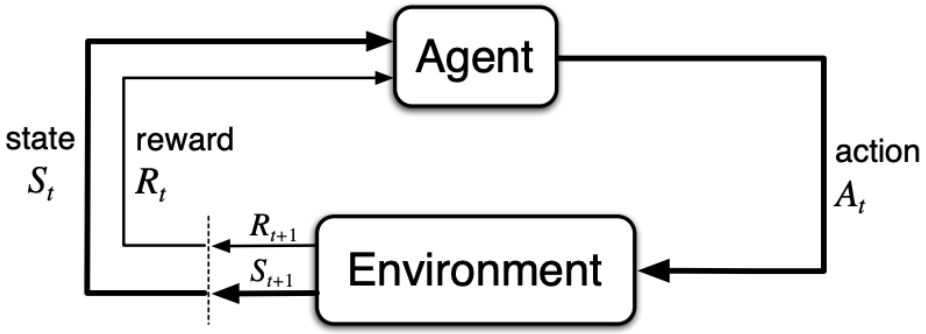


Figure 3.1: Agent-Environment interaction in MDP. [34]

The environment emits the reward signal based on the reward function, r. The state transition to the new state(S_{t+1}) happens based on a stochastic transition function(T). Further, the state transitions have the Markov property:

$$p(s_{t+1}|s_0, a_0, \ldots, s_{t-1}, a_{t-1}, s_t, a_t) = p(s_{t+1}|s_t, a_t)$$

The agent in an MDP takes action based on what action maximizes the reward for the agent. The reward for an action depends not only on the current reward but also on the rewards in future states. The discount factor determines the present value of future rewards.

$$\textit{Reward for action} = r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k\, r_{t+k+1}$$

Definition 2. *A **Markov Decision Process** is defined by the tuple $\langle S, A, \mathrm{T}, \mathrm{r}, \gamma, b_0 \rangle$, where:*

- S is the set of states
- A is the set of allowed actions
- $\mathrm{T} : S \times A \times S \rightarrow [0, 1]S$ is the transition function, denoting the probability of transitioning between the states based on an action $a \in A$.
- $\mathrm{r} : S \times A \rightarrow \mathbb{R}$ is the reward function, denoting the reward signal for taking action $a \in A$ in a state $s \in S$.
- $\gamma < 1$ is the discount factor
- $b_0(s)$ is the start state distribution, denoting the probability of being in a state $s \in S$ at the start.

We use $\pi(a|s) = \Pr(a|s)$ to represent the policy learned by the agent for an MDP. 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.

The optimization problem to solve for the MDP is:

$$\max_{\pi} \sum_{s} b_0(s) V^{\pi}(s) \tag{3.1}$$

$$V^{\pi}(s) = \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t r_t|s, \pi\big] \tag{3.2}$$

Solving this optimization problem will give us the policy, π, learned by the agent. However, a solution obtained by optimizing the primary task alone may result in negative side effects.

3.2 Negative Side Effects

For our discussion, 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 negative side effects.

As discussed in Chapter 1, we define negative side effects as "the undesired effects of an agent's actions that occur in addition to the agent's intended effects when operating in the open world" [30]. 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 3. *Let Λ denote a* partition *of the space of all possible complete trajectories for the given MDP into mutually exclusive sets of* categories *of negative side effects: $\Lambda = \lambda_1 \cup \lambda_2 \cup \ldots \cup \lambda_K$.*

Intuitively, each λ_i defines a category of negative side effects. We assume that having only one set λ_j represents the absence of negative side effects in the corresponding trajectories. Without loss of generality, we assume that a trajectory is associated with a *single category* of negative side effects, which is generally the most severe form of negative side effect that occurs in the trajectory.

When a trajectory $\tau \in \lambda_i$ is encountered during the plan execution, we say that the negative side effect i has been observed. We consider negative side effects that are *non-Markovian* as the category may depend on the entire trajectory, in contrast to Markovian negative side effects that depend on a single state-action pair [27]. Non-Markovian negative side effects are significantly richer than Markovian negative side effects and can model complex negative side effects without the need to expand the state representation.

Practically, it is infeasible to define such partitions accurately. For example, in Figure 1.3, a λ_i may correspond to driving fast through puddles at least k times. There are multiple possible ways in which this can happen; it is infeasible to list all such trajectories. One can only observe some representative samples for different partitions and learn to generalize from such observed data.

3.3 The Optimization Problem

We aim to mitigate negative side effects 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 negative side effect category and E_t denote the category at time t. The optimization problem is noted below.

$$\max_{\pi} \sum_{s} b_0(s) V^{\pi}(s) \tag{3.3}$$

$$V^{\pi}(s) = \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t r_t | s, \pi\big] \tag{3.4}$$

$$\sum_{s} b_0(s)[V^*(s) - V^{\pi}(s)] \leq \zeta \tag{3.5}$$

$$\sum_{t=0}^{\infty} \gamma^t \mathbf{Pr}(E_t = c; \pi) \leq \alpha_c \;\; \forall c = \{1, \ldots, |\mathcal{E}|\} \tag{3.6}$$

Equations (3.3) and (3.4) are part of the standard MDP optimization problem. ζ in equation (3.5), denotes the allowed slack on the agent's primary objective (V^*), obtained by ignoring negative side effects, and equation (3.6) constrains the expected frequency of negative side effect occurrence. The threshold α_c denotes the tolerance for negative side effect category c. The parameters ζ and α_c are typically specified by the user based on their tolerance. They are used to balance the trade-off between optimizing the primary objective and avoiding negative side effects.

The above problem is challenging because the constraints in Equation (3.6) are non-linear and non-convex in policy parameters π. In addition, it is impractical to enumerate all trajectories τ that define a negative side effect category. Therefore, the agent learns to estimate the probability of different categories of negative side effects from historical data.

Chapter 4

Controller to Predict Negative Side Effects

This chapter discusses the controller structure used to represent negative side effects. When the agent has no prior knowledge about the side effects of its actions, it must learn a predictive model of negative side effects. Finite-state controllers(FSCs) provide a simple and compact way of representation. We use them to represent the negative side effects compactly.

We assume that the agent has access to a dataset to learn the finite state controller that contains trajectories for different negative side effect categories $i \in E$. Let $\hat{E}_i$ denote the collection of trajectories for negative side effect 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. We add specifications to the controller to reduce the randomness and make learning easier.

4.1 Finite State Controllers

A finite state controller starts in an initial node ($u_{\mathfrak{s}}$), and for each observation, the controller, based on the current node (u_t) and the observation (σ_t) from the set of observations (Σ), gives an output symbol (c_{t+1}) and changes to a new node (u_{t+1}) from the set of nodes (U).

Definition 4. *A **Finite State Controller** is defined by the tuple $\langle \Sigma, E, U, u_{\mathfrak{s}}, u_{\perp}, \delta, \omega \rangle$, where:*

- Σ is a finite set of observations;
- E is a finite set of output symbols;
- U is a finite set of nodes;
- $u_{\mathfrak{s}}$ as the initial node;

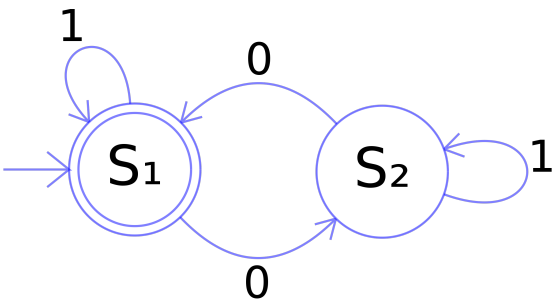


Figure 4.1: Example of a finite state controller. [37]

- $\delta : U \times 2^{\Sigma} \to \Delta U$ is the transition function; and
- $\omega : U \times \Sigma \times U \to E$ is a function that determines the output symbol.

The transition function (δ) is a stochastic function denoting the probability of transitioning between the nodes after receiving an observation $\sigma \in \Sigma$. The controller emits the output symbol ($c_t \in E$) based on the output function (ω).

4.2 The NSE Controller

The existing approaches use a tabular representation for negative side effects. However, this approach suffers from two key limitations:

1. it is not scalable to large problems with Markovian negative side effects; and
2. it cannot represent non-Markovian negative side effects.

To overcome these drawbacks, we propose using a finite state controller to learn about and compactly represent both Markovian and non-Markovian negative side effects.

A finite state controller can compactly summarize the 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 finite state controller nodes, also called the cross-product MDP [16]. This decoupled representation eliminates the need for updating the MDP to represent negative side effects, which may require extensive testing to ensure no new risks are introduced. Furthermore, finite state controllers provide an explainable form for learning negative side effects; empirically, a small dataset was sufficient to learn their structure.

We assume there are K negative side effect categories (or $|\mathcal{E}| = K$). Let $[K]$ denote the set $\{1, \ldots, K\}$.

Definition 5. *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, and 2^{Σ} is a finite set of observations;*
- *$E = [K] \cup \rho$ is a finite set of output symbols that denote various negative side effect categories, with an empty output symbol at intermediate nodes to handle non-Markovian negative side effect;*
- *U is a finite set of nodes;*
- *$u_{\mathfrak{s}}$ as the initial node;*
- *$u_{\perp}$ as the terminal node;*
- *$\delta : U \times 2^{\Sigma} \to \Delta U$ 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*
- *$\omega : U \times 2^{\Sigma} \times U \to \Delta E$ denotes the probability of an NSE category, given the nodes and input symbol.*

We add a terminal node, $u_{\perp}$, and all the trajectories always end in the terminal node, i.e., when an agent reaches the goal state in the MDP, the controller will transition to the terminal node. The output symbol at the terminal node represents the negative side effect category of a trajectory. All intermediate nodes emit a dummy symbol (ρ).

Figure 4.2 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 $L : S \times A \times S \to 2^{\Sigma}$. The labeling function assigns truth values to propositions Σ, given (s, a, s').

Intuitively, the observation σ is a high-level view of the low-level environment states and actions and is important for the generalizability of the learned controller. Such labeling functions have been used previously for learning controllers for POMDPs [22]. Similar to their work, we assume that a labeling function is provided as part of the problem definition.

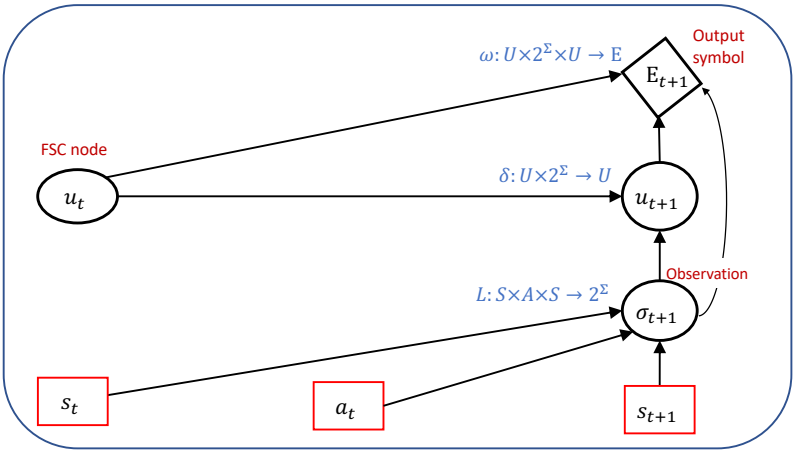


Figure 4.2: 2-Slice dynamic Bayesian net representing the FSC classifier structure. The state-action trajectory (bottom row, red color) is observed; u_t, u_{t+1} denote the controller nodes at time t, $t+1$; σ_{t+1} is the high-level observation received, E_{t+1} is the output symbol representing the observed negative side effect category for the input state-action trajectory. The ω and δ are the key parameters to learn.

Predicting the negative side effect associated with an agent trajectory using a finite state controller 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 negative side effect category associated with the trajectory.

For Markovian negative side effects, each node in the controller may be able to predict the negative side effect category associated with each (s, a, s') experience. For non-Markovian negative side effects, all the states and actions in the trajectory need to be accounted for before determining the negative side effect. Therefore, all intermediate nodes output ρ deterministically, and the terminal node $u_{\perp}$, corresponding to the end of the trajectory, will determine the negative side effect category.

We briefly describe the negative side effect prediction using a finite state controller for the autonomous vehicle domain in Figure 4.3. Let controller nodes track the number of times the autonomous vehicle navigated through a puddle with and without pedestrians nearby, along with speed and whether the

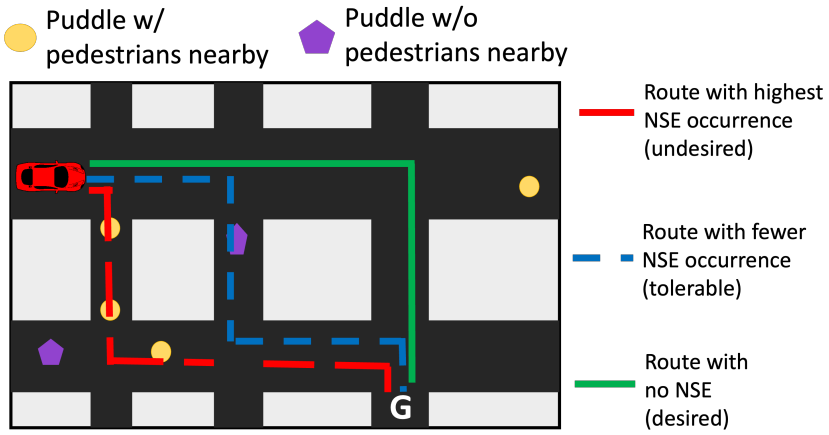


Figure 4.3: Illustration of a non-Markovian negative side effect in a driving domain, in which the negative side effect is associated with the frequency of driving fast through puddles along the route.

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 negative side effect occurs when driving fast through a puddle without nearby pedestrians, and severe negative side effect occurs when driving fast through a puddle with pedestrians nearby. When the autonomous vehicle 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 autonomous vehicle 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 negative side effect.

Let us now consider a *non-Markovian* version where navigating fast through puddles without nearby pedestrians for more than 25% of its trajectory length results in a mild negative side effect, and driving fast through puddles with pedestrians nearby results in severe negative side effect. 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 negative side effect.

Chapter 5

Learning the NSE Controller

In this chapter, we introduce an algorithm to learn the parameters of the NSE controller. Since negative side effects (NSEs) are often discovered after deployment, defining the associated finite state controller during design is impossible. Therefore, the agent must gather information about negative side effects and learn the NSE controller.

5.1 Training Data

Let $\hat{E}_c$ denote the set of state-action trajectories of negative side effect category c. The training data for the controller is of the form $\{(x, y)\}$ where x is the input to the controller 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 negative side effect category associated with $\hat{E}_c$. Such training data can be generated for all $\hat{E}_c$ using various forms of feedback, such as by exploring the environment, from human feedback, or from the past deployment of the system [30]. For our empirical results, we used an ϵ-greedy policy using the optimal primary value function $V^\star$ to collect this data.

5.2 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 NSE controller is shown in Figure 5.1. Observed values are represented using square nodes, and hidden variables using ellipses. Assume τ is a T-step trajectory. In our setting, the node at the

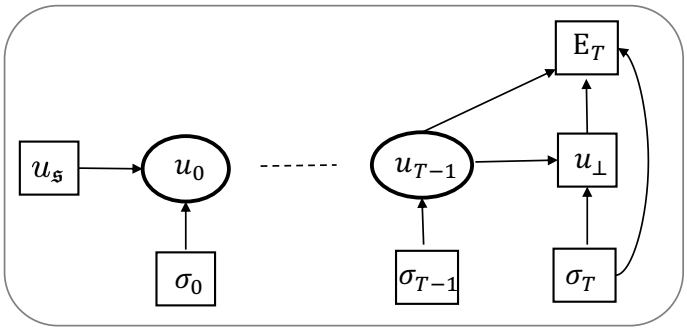


Figure 5.1: Markov chain representing the controller-based classification of a trajectory.

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):

$$\max_{\delta,\omega} p(\sigma_{0:T}, u_{\mathfrak{s}}, u_\perp, E_T \mid \delta, \omega). \tag{5.1}$$

As $u_{0:T-1}$ are hidden variables, we can treat (5.1) as an MLE problem with missing data and use the well-known Expectation-Maximization(EM) algorithms to estimate parameters δ, ω [7]. The EM algorithm has a particularly well-suited structure for exponential family distributions [6] with the MLE often solvable analytically. An exponential family is a set of probability distributions with a probability density function of the form:

$$\log p(x, y \mid \theta) = D(\theta) \cdot \mathcal{T}(x, y) - C(\theta) + B(x, y), \tag{5.2}$$

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:

$$\theta^\star = \arg\max_{\theta' \in \Omega} \big[D(\theta') \cdot \overline{\mathcal{T}(x, y)} - C(\theta')\big]. \tag{5.3}$$

5.3 Exponential Family

We show that the joint distribution for the model in figure 5.1 belongs to the exponential family and characterize its sufficient statistic to formulate the equivalent optimization problem as (5.3). For our case, $\theta = (\delta, \omega)$. Let M denote total nodes in our NSE controller (including terminal and start node), and $[M] = \{1, \ldots, M\}$. Let indices $m, n \in [M]$ be used to index over NSE controller nodes. Using an over-complete representation, we define:

- u_t as M-dimensional one hot vector, with $u_t^m = 1$ means controller node is m at time step 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 step t.
- Let $c \in [|E|]$ index over output symbols. Let E_T be $|E|$-dimensional one hot vector defined analogously to σ_t. $E_T^c = 1$ indicates output symbol c is emitted at time step T.

To learn the **NSE controller**, we must learn the parameters δ and 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 negative side effects, 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 the last node was m, the current observation is i, and the current node is $u_\perp$. We use shorthand $\omega(c|m, i)$ by omitting $u_\perp$. Let $\boldsymbol{u} = (u_s, u_{0:T-1}, u_\perp)$, and $\boldsymbol{\sigma} = \sigma_{0:T}$ denote the complete data.

5.3.1 NSE controller's relation to Exponential Family

Theorem 1. *Let $\boldsymbol{u} = (u_s, u_{0:T-1}, u_\perp)$, $\boldsymbol{\sigma} = \sigma_{0:T}$. The complete data distribution $p(\boldsymbol{u}, \boldsymbol{\sigma}, E_T; \delta, \omega)$ for the model in Figure 5.1 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:*

$$T(\boldsymbol{u}, \boldsymbol{\sigma}, E_T) \leftarrow [u_0^m \sigma_0^i]\ \forall m, i;\ [u_{T-1}^m \sigma_T^i]\ \forall m, i;$$
$$\Big[\sum_{t=0}^{T-2} u_t^m u_{t+1}^n \sigma_{t+1}^i\Big]\ \forall m, n, i;\quad [E_T^c u_{T-1}^m \sigma_T^i]\ \forall c, m, i$$

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

- $C(\delta, \omega) \leftarrow 0$

Proof. Transition probability:

$$p(u_{t+1}|u_t, \sigma_{t+1}) = \prod_{m,n,i=1}^{|U|,|U|,|\Sigma|} [\delta(n|m,i)]^{u_t^m u_{t+1}^n \sigma_{t+1}^i} \tag{5.4}$$

Transition at the first step:

$$p(u_0|u_s, \sigma_0) = \prod_{m,i=1}^{|U|,|\Sigma|} [\delta(m|u_s,i)]^{u_0^m \sigma_0^i} \tag{5.5}$$

Transition at last step:

$$p(u_\perp|u_{T-1}, \sigma_T) = \prod_{m,i=1}^{|U|,|\Sigma|} [\delta(u_\perp|m,i)]^{u_{T-1}^m \sigma_T^i} \tag{5.6}$$

Emission Probability:

$$p(\Gamma_T|u_{T-1}, \sigma_T) = \prod_{a,m,i=1}^{|\Gamma|,|U|,|\Sigma|} [\omega(a|m,i)]^{\Gamma_T^a u_{T-1}^m \sigma_T^i} \tag{5.7}$$

Explanation Consider an NSE controller transitions from node 1 to node 2 on an observation 0 at time step $t+1$, then

- $u_t^m = 1$ if $m = 1$ and $u_t^m = 0$ if $m \neq 1$
- $u_{t+1}^n = 1$ if $n = 2$ and $u_{t+1}^n = 0$ if $n \neq 2$
- $\sigma_{t+1}^i = 1$ if $i = 0$ and $\sigma_{t+1}^i = 0$ if $i \neq 0$

This gives us $p(u_{t+1}|u_t, \sigma_{t+1}) = \delta(2|1,0)$, which matches the transition probability of the NSE controller. Similarly, equations (5.5), (5.6), and (5.7) solve to match the transition and emission probabilities of the NSE controller.

Now, we show that the distribution in equation (5.1) belongs to the exponential family by taking its log and using the transition and emission probabilities to expand it:

$$
\begin{aligned}
&\log p(u_{0:T}, \sigma_{0:T}, E_T \mid \delta, \omega) \\
&= \log \left(p(u_0|u_{\mathfrak{s}}, \sigma_0) \times \prod_{t=0}^{T-2} p(u_{t+1}|u_t, \sigma_{t+1}) \times p(u_\perp|u_{T-1}, \sigma_T) \times p(E_T|u_{T-1}, \sigma_T) \times \prod_{t=0}^{T} p(\sigma_t) \right) \\
&= \log \left(\prod_{m,i=1}^{|U|,2^{|\Sigma|}} [\delta(m|u_{\mathfrak{s}}, i)]^{u_0^m \sigma_0^i} \times \prod_{t=0}^{T-2} \prod_{m,n,i=1}^{|U|,|U|,2^{|\Sigma|}} [\delta(n|m,i)]^{u_t^m u_{t+1}^n \sigma_{t+1}^i} \right. \\
&\qquad \left. \times \prod_{m,i=1}^{|U|,2^{|\Sigma|}} [\delta(u_\perp|m,i)]^{u_{T-1}^m \sigma_T^i} \times \prod_{c,m,i=1}^{|E|,|U|,2^{|\Sigma|}} [\omega(c|m,i)]^{E_T^c u_{T-1}^m \sigma_T^i} \times \prod_{t=0}^{T} p(\sigma_t) \right) \\
&= \sum_{m,i=1}^{|U|,2^{|\Sigma|}} log[\delta(m|u_{\mathfrak{s}}, i)]^{u_0^m \sigma_0^i} + \sum_{t=0}^{T-2} \sum_{m,n,i=1}^{|U|,|U|,2^{|\Sigma|}} log[\delta(n|m,i)]^{u_t^m u_{t+1}^n \sigma_{t+1}^i} + \\
&\qquad + \sum_{m,i=1}^{|U|,2^{|\Sigma|}} \log[\delta(u_\perp|m,i)]^{u_{T-1}^m \sigma_T^i} + \sum_{c,m,i=1}^{|E|,|U|,2^{|\Sigma|}} \log[\omega(c|m,i)]^{E_T^c u_{T-1}^m \sigma_T^i} + \sum_{t=0}^{T} \log p(\sigma_t) \\
&= \sum_{m,i=1}^{|U|,2^{|\Sigma|}} u_0^m \sigma_0^i log[\delta(m|u_{\mathfrak{s}}, i)] + \sum_{t=0}^{T-2} \sum_{m,n,i=1}^{|U|,|U|,2^{|\Sigma|}} u_t^m u_{t+1}^n \sigma_{t+1}^i log[\delta(n|m,i)] \\
&\qquad + \sum_{m,i=1}^{|U|,2^{|\Sigma|}} u_{T-1}^m \sigma_T^i \log[\delta(u_\perp|m,i)] + \sum_{c,m,i=1}^{|E|,|U|,2^{|\Sigma|}} E_T^c u_{T-1}^m \sigma_T^i \log[\omega(c|m,i)] + \sum_{t=0}^{T} \log p(\sigma_t) \\
&= \sum_{m,i=1}^{|U|,2^{|\Sigma|}} \left(u_0^m \sigma_0^i\right) log[\delta(m|u_{\mathfrak{s}}, i)] + \sum_{m,n,i=1}^{|U|,|U|,2^{|\Sigma|}} \left(\sum_{t=0}^{T-2} u_t^m u_{t+1}^n \sigma_{t+1}^i \right) log[\delta(n|m,i)] \\
&\qquad + \sum_{m,i=1}^{|U|,2^{|\Sigma|}} \left(u_{T-1}^m \sigma_T^i\right) \log[\delta(u_\perp|m,i)] + \sum_{c,m,i=1}^{|E|,|U|,2^{|\Sigma|}} \left(E_T^c u_{T-1}^m \sigma_T^i\right) \log[\omega(c|m,i)] + \sum_{t=0}^{T} \log p(\sigma_t)
\end{aligned}
$$

From above, we have:

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

$$
\begin{aligned}
&T(\boldsymbol{u}, \boldsymbol{\sigma}, E_T) \leftarrow [u_0^m \sigma_0^i] \; \forall m, i; \; [u_{T-1}^m \sigma_T^i] \; \forall m, i; \\
&\Big[\sum_{t=0}^{T-2} u_t^m u_{t+1}^n \sigma_{t+1}^i \Big] \; \forall m, n, i; \quad [E_T^c u_{T-1}^m \sigma_T^i] \; \forall c, m, i
\end{aligned}
$$

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

- $C(\delta, \omega) \leftarrow 0$

Therefore, $\log p(u_{0:T}, \sigma_{0:T}, E_T \mid \delta, \omega)$ belongs to an exponential family. □

5.3.2 Optimization Problem for Learning Controller Parameters

Using the above terms, the equivalent optimization problem to (5.3) for NSE controller learning is given below. To avoid clutter, we show terms only for a single training data point; the final optimization problem involves the summation of analogous terms for all the training data points.

$$\max_{\delta,\omega} \sum_{m,i} \mathbb{E}\big[u_0^m \sigma_0^i\big] \log[\delta(m|u_{\mathfrak{s}}, i)] + \sum_{m,n,i} \mathbb{E}\big[\sum_{t=0}^{T-2} u_t^m u_{t+1}^n \sigma_{t+1}^i\big] \times \log[\delta(n|m,i)] +$$
$$\sum_{m,i} \mathbb{E}\big[u_{T-1}^m \sigma_T^i\big] \log[\delta(u_\perp|m,i)] + \sum_{c,m,i} \mathbb{E}\big[E_T^c u_{T-1}^m \sigma_T^i\big] \log[\omega(c|m,i)] \tag{5.8}$$

$$\sum_{n\in[M]} \delta(n|m,i) = 1 \quad \forall m \in [M], i \in [2^{|\Sigma|}] \tag{5.9}$$

$$\sum_{c\in[|E|]} \omega(c|m,i,u_\perp) = 1 \quad \forall m \in [M], i \in [2^{|\Sigma|}] \tag{5.10}$$

$$\delta(u_\perp|u_\perp,i) = 1 \; \forall i \in [2^{|\Sigma|}] \tag{5.11}$$

$$\delta(u_{\mathfrak{s}}|m,i) = 0 \; \forall m \in [M], \forall i \in [2^{|\Sigma|}] \tag{5.12}$$

$$\omega(\rho|u_\perp,i,u_\perp) = 1 \; \forall i \in [2^{|\Sigma|}] \tag{5.13}$$

The constraints (5.9) and (5.10) are standard probability normalization constraints. Constraint (5.11) ensures that $u_\perp$ is an absorbing node without any outgoing transitions. Constraint (5.12) ensures that there is no incoming transition to the starting node $u_{\mathfrak{s}}$. Constraint (5.13), along with constraint (5.11), 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 the above program is often much lower than exponential in the number of propositions.

Forward-backward algorithm, similar to the well-known Baum-Welch algorithm [6] adapted to our setting, can be used to calculate expected sufficient statistics. We can also use the Karush–Kuhn–Tucker

(KKT) conditions [5] to solve the problem analytically to obtain improved estimates of δ and ω parameters.

5.4 Forward-Backward Algorithm

The Forward-Backward Algorithm is used to calculate the probability of being in controller node m at time t given a sequence of input symbols $\sigma_{1:T}$ assuming T is the length of the trajectory.

Forward Step: We calculate $\mu_m(t)$ as the probability of being in controller node m at time t given a sequence of input symbols $\sigma_{1:t}$.

$$
\begin{aligned}
\mu_m(t+1) &= P(u_{t+1} = m | \sigma_{1:t}, \delta, \omega) \\
&= \sum_{u_0,\ldots,u_t} \delta(u_0|u_{\mathfrak{s}}, \sigma_0) \times \prod_{k=0}^{t-1} \delta(u_{k+1}|u_k, \sigma_{k+1}) \times \delta(m|u_t, \sigma_t) \\
&= \sum_{u_t \in U} \sum_{u_0,\ldots,u_{t-1}} \delta(u_0|u_{\mathfrak{s}}, \sigma_0) \times \prod_{k=0}^{t-1} \delta(u_{k+1}|u_k, \sigma_{k+1}) \times \delta(m|u_t, \sigma_t) \\
&= \sum_{u_t \in U} \left(\sum_{u_0,\ldots,u_{t-1}} \delta(u_0|u_{\mathfrak{s}}, \sigma_0) \times \prod_{k=0}^{t-2} \delta(u_{k+1}|u_k, \sigma_{k+1}) \times \delta(u_t|u_{t-1}, \sigma_t) \right) \times \delta(m|u_t, \sigma_t) \\
&= \sum_{u_t \in U} \mu_{u_t}(t) \delta(m|u_t, \sigma_t) \\
&= \sum_{n \in U} \mu_n(t) \delta(m|n, \sigma_t)
\end{aligned}
$$

Therefore, we have:

$$
\begin{aligned}
\mu_m(0) &= \delta(m|u_{\mathfrak{s}}, \sigma_0) \\
\mu_m(t+1) &= \sum_{n \in U} \mu_n(t) \delta(m|n, \sigma_t)
\end{aligned}
$$

Backward Step: We calculate $\beta_m(t)$ as the probability of being in controller node m at time t given a sequence of input symbols $\sigma_{t+1:T}$ and output symbol c.

$$
\begin{aligned}
\beta_m(t) &= P(u_t = m | \sigma_{t+1:T}, c, \delta, \omega) \\
&= \sum_{u_{t+1},\ldots,u_{T-1}} \delta(u_{t+1}|m, \sigma_{t+1}) \times \prod_{k=t+1}^{T-2} \delta(u_{k+1}|u_k, \sigma_{k+1}) \times \delta(u_\perp|u_{T-1}, \sigma_T) \times \omega(c|u_{T-1}, \sigma_T) \\
&= \sum_{u_{t+1}} \sum_{u_{t+2},\ldots,u_{T-1}} \delta(u_{t+1}|m, \sigma_{t+1}) \times \prod_{k=t+1}^{T-2} \delta(u_{k+1}|u_k, \sigma_{k+1}) \times \delta(u_\perp|u_{T-1}, \sigma_T) \times \omega(c|u_{T-1}, \sigma_T)
\end{aligned}
$$

$$= \sum_{u_{t+1}} \delta(u_{t+1}|m, \sigma_{t+1}) \times$$
$$\left(\sum_{u_{t+2},\ldots,u_{T-1}} \delta(u_{t+2}|u_{t+1}, \sigma_{t+2}) \prod_{k=t+2}^{T-2} \delta(u_{k+1}|u_k, \sigma_{k+1}) \times \delta(u_\perp|u_{T-1}, \sigma_T) \times \omega(c|u_{T-1}, \sigma_T) \right)$$
$$= \sum_{u_{t+1} \in U} \delta(u_{t+1}|m, \sigma_{t+1}) \beta_{u_{t+1}}(t+1)$$
$$= \sum_{n \in U} \beta_n(t+1) \delta(j|m, \sigma_{t+1})$$

Therefore, we have:

$$\beta_m(T-1) = \delta(u_\perp|m, \sigma_T)\omega(c|m, \sigma_T)$$
$$\beta_m(t) = \sum_{j \in U} \beta_n(t+1)\delta(n|m, \sigma_{t+1})$$

We calculate $\upsilon_m(t)$ as probability of being in controller node m at time t given a sequence of input symbols $\sigma_{1:T}$ and output symbol c:

$$\upsilon_m(t) = \frac{\mu_m(t)\beta_m(t)}{\sum_{n \in U} \mu_n(t)\beta_n(t)}$$

We calculate $\xi_{mn}(t)$ as probability of being in controller node m at time t and in controller node n at time $t+1$ given a sequence of input symbols $\sigma_{1:T}$ and output symbol c:

$$\xi_{mn}(t) = \frac{\mu_m(t)\delta(n|m, \sigma_{t+1})\beta_n(t+1)}{\sum_{o \in U} \sum_{p \in U} \mu_o(t)\delta(p|o, \sigma_{t+1})\beta_p(t+1)}$$

We have,

$$\sum_{n=1}^{|U|} \xi_{mn}(t) = \upsilon_m(t)$$

5.4.1 Computational Complexity

The computational complexity of the forward-backward algorithm is $O(RM^2T)$ assuming there are R training trajectories of length T, and the FSC has M nodes. Empirically, moderately sized controllers ($M = 10, 11$) were sufficient to accurately classify trajectories.

5.4.2 Solving for Controller Parameters

We can solve the optimization problem in section 5.3.2 using KKT conditions.

The Lagrangian is given as:

$$L(\delta,\omega,\lambda,v) = \sum_{m,i}\mathbb{E}\big[u_0^m\sigma_0^i\big]\log[\delta(m|u_{\mathfrak{s}},i)] + \sum_{m,n,i}\mathbb{E}\big[\sum_{t=0}^{T-2}u_t^m u_{t+1}^n\sigma_{t+1}^i\big]\times\log[\delta(n|m,i)]$$
$$+\sum_{m,i}\mathbb{E}\big[u_{T-1}^m\sigma_T^i\big]\log[\delta(u_\perp|m,i)] + \sum_{c,m,i}\mathbb{E}\big[E_T^c u_{T-1}^m\sigma_T^i\big]\log[\omega(c|m,i)]$$
$$+\sum_{m,i=1}^{|U|,2^{|\Sigma|}}\left(\lambda_{m,i}(\sum_{n=1}^{|U|}\delta(n|m,i)-1)+v_{m,i}(\sum_{c=1}^{|E|}\omega(c|m,i)-1)\right) \tag{5.14}$$

KKT conditions are given as:

$$\nabla L(\delta,\omega,\lambda,v) = 0 \tag{5.15}$$

$$\sum_{n=1}^{|U|}\delta(n|m,i) = 1 \quad \forall m\in U, i\in 2^{|\Sigma|} \tag{5.16}$$

$$\sum_{c=1}^{|E|}\omega(c|m,I,u_\perp) = 1 \quad \forall m\in U, i\in 2^{|\Sigma|} \tag{5.17}$$

Differentiating Langrangian(Equation 5.14) with respect to each variable:

$$\frac{\partial L}{\partial\delta(m|u_{\mathfrak{s}},i)} = \frac{\big(u_0^m\sigma_0^i\big)}{\delta(m|u_{\mathfrak{s}},i)} + \lambda_{u_{\mathfrak{s}},i} = 0$$
$$\delta(m|u_{\mathfrak{s}},i) = -\frac{\big(u_0^m\sigma_0^i\big)}{\lambda_{u_{\mathfrak{s}},i}} \tag{5.18}$$

$$\frac{\partial L}{\partial\delta(n|m,i)} = \frac{\Big(\sum_{t=0}^{T-2}u_t^m u_{t+1}^n\sigma_{t+1}^i\Big)}{\delta(n|m,i)} + \lambda_{m,i} = 0$$
$$\delta(n|m,i) = -\frac{\Big(\sum_{t=0}^{T-2}u_t^m u_{t+1}^n\sigma_{t+1}^i\Big)}{\lambda_{m,i}} \tag{5.19}$$

$$\frac{\partial L}{\partial\delta(u_\perp|m,i)} = \frac{\big(u_{T-1}^m\sigma_T^i\big)}{\delta(u_\perp|m,i)} + \lambda_{m,i} = 0$$
$$\delta(u_\perp|m,i) = -\frac{\big(u_{T-1}^m\sigma_T^i\big)}{\lambda_{m,i}} \tag{5.20}$$

$$\frac{\partial L}{\partial\omega(c|m,i)} = \frac{\big(E_T^c u_{T-1}^m\sigma_T^i\big)}{\omega(c|m,i)} + v_{m,i} = 0$$
$$\omega(c|m,i) = -\frac{\big(E_T^c u_{T-1}^m\sigma_T^i\big)}{v_{m,i}} \tag{5.21}$$

Using Equation 3:

$$\sum_{m=1}^{|U|} \delta(m|u_{\mathfrak{s}}, i) = 1$$

$$\sum_{m=1}^{|U|} -\frac{\left(u_0^m \sigma_0^i\right)}{\lambda_{u_{\mathfrak{s}},i}} = 1$$

$$\lambda_{u_{\mathfrak{s}},i} = -\sum_{m=1}^{|U|} \left(u_0^m \sigma_0^i\right)$$

$$\lambda_{u_{\mathfrak{s}},i} = \sigma_0^i \sum_{m=1}^{|U|} u_0^m = \sigma_0^i \tag{5.22}$$

$$\sum_{n=1}^{|U|} \delta(n|m, i) = 1$$

$$\sum_{n=1}^{|U|} -\frac{\left(\sum_{t=0}^{T-2} u_t^m u_{t+1}^n \sigma_{t+1}^i\right)}{\lambda_{m,i}} - \frac{\left(u_{T-1}^m \sigma_T^i\right)}{\lambda_{m,i}} = 1$$

$$\lambda_{m,i} = -\sum_{n=1}^{|U|} \left(\sum_{t=0}^{T-2} u_t^m u_{t+1}^n \sigma_{t+1}^i\right) - \left(u_{T-1}^m \sigma_T^i\right) \tag{5.23}$$

Using Equation 4:

$$\sum_{c=1}^{|E|} \omega(c|m, i) = 1$$

$$\sum_{c=1}^{|E|} -\frac{\left(E_T^c u_{T-1}^m \sigma_T^i\right)}{v_{m,i}} = 1$$

$$v_{m,i} = -\sum_{c=1}^{|E|} \left(E_T^c u_{T-1}^m \sigma_T^i\right) \tag{5.24}$$

Substituting values from 9, 10 and 11 to determine the parameter values:

$$\delta(m|u_{\mathfrak{s}}, i) = \frac{u_0^m \sigma_0^i}{\sigma_0^i}$$

$$\delta(n|m, i) = \frac{\sum_{t=0}^{T-2} u_t^m u_{t+1}^n \sigma_{t+1}^i}{\sum_{o=1}^{|U|} \sum_{t=0}^{T-2} u_t^m u_{t+1}^o \sigma_{t+1}^i + u_{T-1}^m \sigma_T^i}$$

$$\delta(u_{\perp}|m, i) = \frac{u_{T-1}^m \sigma_T^i}{\sum_{n=1}^{|U|} \sum_{t=0}^{T-2} u_t^m u_{t+1}^n \sigma_{t+1}^i + u_{T-1}^m \sigma_T^i}$$

$$\omega(c|m, i) = \frac{E_T^c u_{T-1}^m \sigma_T^i}{\sum_{b=1}^{|E|} E_T^b u_{T-1}^m \sigma_T^i}$$

The value of sufficient variables from the Forward-Backward Algorithm:

$$u_0^m \sigma_0^i = \mathbb{I}[\sigma_0 = i] \upsilon_m(0) \sum_{t=0}^{T-2} u_t^m u_{t+1}^n \sigma_{t+1}^i \qquad = \sum_{t=0}^{T-2} \mathbb{I}[\sigma_{t+1} = i] \xi_{mn}(t)$$

$$u_{T-1}^m \sigma_T^i = \mathbb{I}[\sigma_T = i] \upsilon_m(T-1)$$

$$E_T^c u_{T-1}^m \sigma_T^i = \mathbb{I}[E_T = c] \mathbb{I}[\sigma_T = i] \upsilon_m(T-1)$$

Final update for ω and δ values:

$$\delta(m|u_{\mathfrak{s}}, i) = \frac{\mathbb{I}[\sigma_0 = i] \upsilon_m(0)}{\mathbb{I}[\sigma_0 = i]}$$

$$\delta(n|m, i) = \frac{\sum_{t=0}^{T-2} \mathbb{I}[\sigma_{t+1} = i] \xi_{mn}(t)}{\sum_{o=1}^{|U|} \sum_{t=0}^{T-2} \mathbb{I}[\sigma_{t+1} = i] \xi_{mo}(t) + \mathbb{I}[\sigma_T = i] \upsilon_m(T-1)}$$

$$\delta(u_\perp|m, i) = \frac{\mathbb{I}[\sigma_T = i] \upsilon_m(T-1)}{\sum_{n=1}^{|U|} \sum_{t=0}^{T-2} \mathbb{I}[\sigma_{t+1} = i] \xi_{mn}(t) + \mathbb{I}[\sigma_T = i] \upsilon_m(T-1)}$$

$$\omega(c|m, i) = \frac{\mathbb{I}[E_T = c] \mathbb{I}[\sigma_T = i] \upsilon_m(T-1)}{\sum_{b=1}^{|E|} \mathbb{I}[E_T = b] \mathbb{I}[\sigma_T = i] \upsilon_m(T-1)}$$

Update for ω and δ values when there are multiple trajectories. Let the number of trajectories be R,

$$\delta(m|u_{\mathfrak{s}}, i) = \frac{\sum_{r=1}^{R} \mathbb{I}[\sigma_{r,0} = i] \upsilon_{mr}(0)}{\sum_{r=1}^{R} \mathbb{I}[\sigma_{r,0} = i]}$$

$$\delta(n|m, i) = \frac{\sum_{r=1}^{R} \sum_{t=1}^{T-2} \mathbb{I}[\sigma_{r,t+1} = i] \xi_{mnr}(t)}{\sum_{o=1}^{|U|} \sum_{r=1}^{R} \sum_{t=0}^{T-2} \mathbb{I}[\sigma_{r,t+1} = i] \xi_{mor}(t) + \sum_{r=1}^{R} \mathbb{I}[\sigma_{r,T} = i] \upsilon_{mr}(T-1)}$$

$$\delta(u_\perp|m, i) = \frac{\sum_{r=1}^{R} \mathbb{I}[\sigma_{r,T} = i] \upsilon_{mr}(T-1)}{\sum_{o=1}^{|U|} \sum_{r=1}^{R} \sum_{t=0}^{T-2} \mathbb{I}[\sigma_{r,t+1} = i] \xi_{mor}(t) + \sum_{r=1}^{R} \mathbb{I}[\sigma_{r,T} = i] \upsilon_{mr}(T-1)}$$

$$\omega(c|m, i) = \frac{\sum_{r=1}^{R} \mathbb{I}[\Gamma_{r,T} = c] \mathbb{I}[\sigma_{r,T} = i] \upsilon_{mr}(T-1)}{\sum_{b=1}^{|\Gamma|} \sum_{r=1}^{R} \mathbb{I}[\Gamma_{r,T} = b] \mathbb{I}[\sigma_{r,T} = i] \upsilon_{mr}(T-1)}$$

5.5 Expectation-Maximization Algorithm for Learning NSE Controller

During random initialization, we have:

$$\delta(u_\perp|u_\perp, i) = 1 \; \forall i \in [2^{|\Sigma|}]$$

$$\delta(u_{\mathfrak{s}}|m, i) = 0 \; \forall m \in [M], \forall i \in [2^{|\Sigma|}]$$

$$\omega(\phi|u_\perp, i) = 1 \; \forall i \in [2^{|\Sigma|}]$$

These terms are not updated during the EM iterations as they are hard constraints in the problem stated in section 5.3.2.

Function `EM(`R`)`**:**

$b\_\delta, b\_\omega \leftarrow \phi$ `//store best parameters over 10 trials`

for $itr \in range(10)$ **do**

`//10 denotes the number of FSC learning runs`

$\delta, \omega \leftarrow$ Random Initialization

$\nabla \leftarrow 1$

$o\_obj = objective(R, \delta, \omega)$

while $\nabla > 0.005$ **do**

$$\delta(m|u_{\mathfrak{s}}, i) = \frac{\sum_{r=1}^{R} \mathbb{I}[\sigma_{r,0}=i]\gamma_{mr}(0)}{\sum_{r=1}^{R} \mathbb{I}[\sigma_{r,0}=i]}$$

$$\delta(n|m, i) = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T-2} \mathbb{I}[\sigma_{r,t+1}=i]\xi_{mnr}(t)}{\sum_{o=1}^{|U|}\sum_{r=1}^{R}\sum_{t=0}^{T-2} \mathbb{I}[\sigma_{r,t+1}=i]\xi_{mor}(t)+\sum_{r=1}^{R} \mathbb{I}[\sigma_{r,T}=i]\gamma_{mr}(T-1)}$$

$$\delta(u_{\perp}|m, i) = \frac{\sum_{r=1}^{R} \mathbb{I}[\sigma_{r,T}=i]\gamma_{mr}(T-1)}{\sum_{o=1}^{|U|}\sum_{r=1}^{R}\sum_{t=0}^{T-2} \mathbb{I}[\sigma_{r,t+1}=i]\xi_{mor}(t)+\sum_{r=1}^{R} \mathbb{I}[\sigma_{r,T}=i]\gamma_{mr}(T-1)}$$

$$\omega(c|m, i) = \frac{\sum_{r=1}^{R} \mathbb{I}[\Gamma_{r,T}=c]\mathbb{I}[\sigma_{r,T}=i]\gamma_{mr}(T-1)}{\sum_{b=1}^{|\Gamma|}\sum_{r=1}^{R} \mathbb{I}[\Gamma_{r,T}=b]\mathbb{I}[\sigma_{r,T}=i]\gamma_{mr}(T-1)}$$

$n\_obj = objective(R, \delta, \omega)$

$\nabla = \mid n\_obj - o\_obj \mid$

$o\_obj = n\_obj$

$b\_\delta, b\_\omega \leftarrow best((b\_\delta, b\_\omega), (\delta, \omega))$ `//compare accuracy over the test set`

return b_δ, b_ω

5.6 Experimental Setup

We evaluate the effectiveness of our EM algorithm-based approach in learning and predicting the negative side effects. We assume Markov state representation for the primary objective. In the interest of clarity, we test with three negative side effect categories: *mild NSE, severe NSE,* and *no NSE*. Each action/trajectory can result in a mild, severe, or no NSE.

5.6.1 Boxpushing Domain

In the boxpushing domain, the agent aims to push a box as quickly as possible to a goal location [26]. 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's position, b is a Boolean variable indicating whether the agent is pushing the box, c indicates the

type of the surface: rug or plain. The agent can move in all four directions(up, down, left, or right), each costing $+1$. The agent can also load the box with the 'pick up box' action, which costs $+2$ and wrap the box with a protective sheet using the 'wrap box' action, which costs $+5$. The '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.

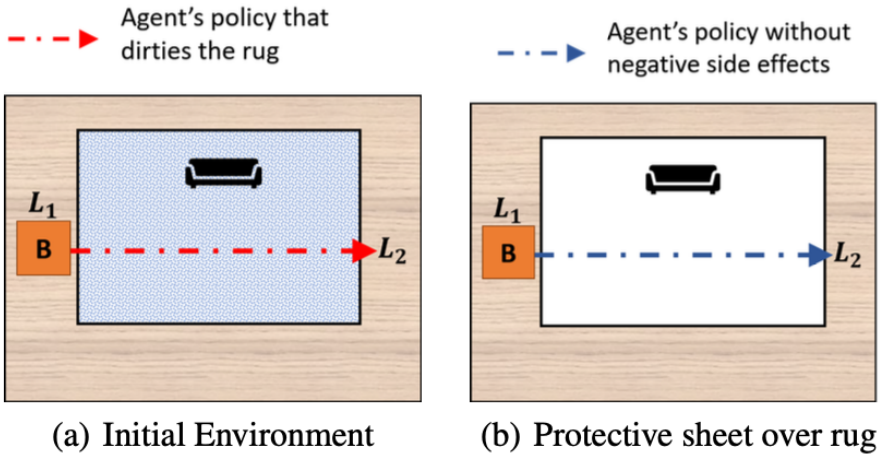


Figure 5.2: Example configurations for boxpushing domain: (a) denotes the initial setting in which the agent dirties the rug when pushing the box over it; (b) denotes a modification that avoids the negative side effects. [29]

Markovian negative side effect occurs when the agent pushes the box over the rug. In *Non-Markovian negative side effect*, the negative side effects are mild when $1-25\%$ rug area is dirtied and severe if more than 25% is dirtied when the agent completes its task. We experiment with grid size 15×15, same as in previous work [26].

5.6.2 Navigation Domain

Our second domain is autonomous vehicle navigation (Figure 4.3), where the autonomous vehicle aims to navigate quickly to a goal location [26, 38, 25]. The autonomous vehicle can move in all four directions(up, down, left, or right) and navigate at two speeds: slow and fast. Driving slow costs $+2$, and driving fast costs $+1$. Each state is represented by $\langle x, y,$ speed, pedestrian, puddle$\rangle$. Pedestrian and puddle are Boolean variables. The autonomous vehicle's move actions succeed with probability 0.9 or

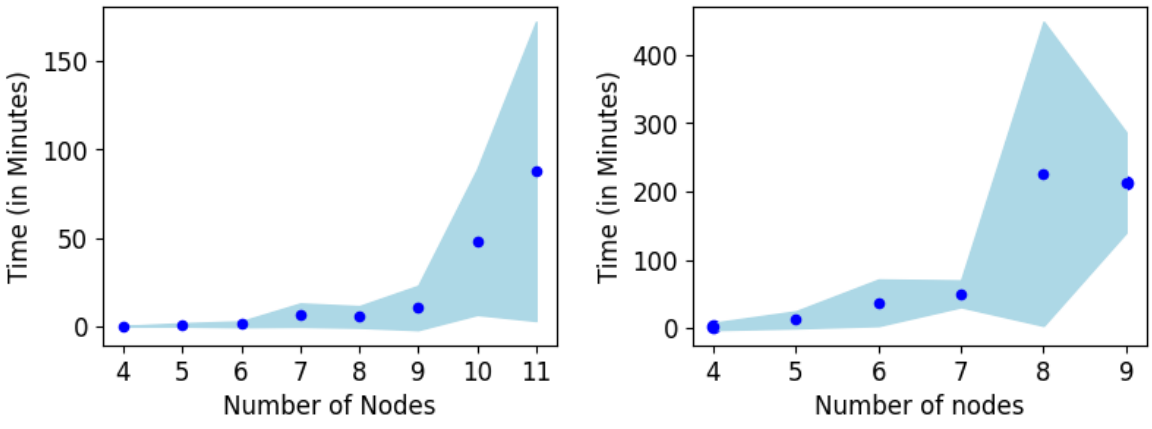
fail with probability 0.1 and slide to a neighboring cell. The agent can transition between any speeds deterministically.

Markovian negative side effect occurs when driving fast through puddles, with or without pedestrians in the vicinity. *Non-Markovian negative side effect*, the negative side effects are mild when the autonomous vehicle drives fast through puddles, without pedestrians in the vicinity, for more than 25% of its route. Driving fast through puddles with pedestrians nearby results in severe NSE. We experiment with grid size 15×15.

5.7 Results

We evaluate the effectiveness of our approach in learning an NSE controller to predict negative side effects using F-1 scores for each negative side effect category and overall prediction accuracy, as we vary the controller size (Table 5.1). 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 the navigation domain since the trajectories are relatively longer and the negative side effect condition is more complex.

5.7.1 Training Time



(a) Boxpushing

(b) Navigation

Figure 5.6: Average training time, with standard deviation, for learning controllers of various sizes.

While the accuracy may improve as we increase the controller size, it also increases the training time (Fig. 5.6). Hence, we choose the smallest size that achieves comparable performance across negative side effect categories and $\sim 90\%$ accuracy.

5.7.2 F-1 Score

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

Based on the results in Table 5.1, we use a controller size of eight nodes for the boxpushing domain and seven nodes for the navigation domain.

Chapter 6

CASP: Controller-Assisted Safe Planning

We will now present a framework to mitigate the negative side effects. The NSE controller can classify the seen and unseen trajectories into various categories of negative side effects. The agent can integrate this controller with the original MDP using the dual LP formulation to mitigate the negative side effects.

6.1 The Optimisation Problem using Dual LP

We now show how the learned NSE controller can be integrated into the dual LP formulation [2]. The optimization formulation we develop approximates the problem (equation (3.4)) as the learned NSE controller may not be fully accurate. We first define the cross-product MDP over the joint space of NSE controller nodes and MDP states, $U \times S$ [17]. We also develop additional constraints to consider negative side effect limits in equation (3.6). The transition function over the cross-product MDP's state space is:

$$P(u', s'|u, s, a) = \mathrm{T}(s, a, s')\delta(u'|u, \sigma = L(s, a, s')) \tag{6.1}$$

where $L(\cdot)$ is the labeling function as discussed in Chapter 4.

The reward function remains the same as $r(s, a)$, unaffected by the controller state. The probability of the negative side effect $E_t = c$:

$$P(E_t = c|u_{t-1}, u_t = u_\perp, s_{t-1}, a_{t-1}, s_t) = \omega\big(c|u_{t-1}, u_\perp, \sigma_t = L(s_{t-1}, a_{t-1}, s_t)\big). \tag{6.2}$$

As we associate a single negative side effect with each trajectory, constraint (5.13) ensures that a non-null negative side effect category is predicted only when the controller reaches the terminal node, $u_\perp$, for the first time.

The dual LP for the cross-product MDP incorporating negative side effect constraints is given in Table 6.1. The structure and interpretation of this dual LP are similar to the standard dual LP for

MDPs [19], 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 the controller state is u, the world state is s, and the action taken is a (represented by 'dual LP flow constraints'). $b_0(u, s)$ in equation (6.4), denotes the probability of starting in controller state u and world state s.

$$\max_{\{y(\cdot)\}} \sum_{u,s,a} r(s,a)y(u,s,a) \tag{6.3}$$

//Dual LP flow constraints

$$\sum_{a} y(u',s',a) = b_0(u',s') + \gamma \sum_{u,s,a} P(u',s'|u,s,a)y(u,s,a) \forall (u',s') \tag{6.4}$$

$$y(u,s,a) \geq 0 \quad \forall (u,s,a) \tag{6.5}$$

//NSE frequency computation

$$y(c) = \sum_{u,s,a} y(u,s,a) \sum_{s'} P(u' = u_\perp, s'|u,s,a) \times \omega(c|u, u_\perp, \sigma = L(s,a,s')) \;\; \forall c \in E \tag{6.6}$$

//NSE satisfaction constraints

$$y(c) \leq \alpha_c \;\; \forall c \in E \tag{6.7}$$

//Primary objective slack

$$\sum_{s} b_0(s)V^*(s) - \sum_{u,s,a} r(s,a)y(u,s,a) \leq \zeta \tag{6.8}$$

Table 6.1: Dual Linear Program for Safe Policy Optimization

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 (6.6)-(6.8) are the major differences from the standard dual LP. Constraint (6.6) compute the probability of different negative side effects $c \in E$ as per our learned controller. Constraint (6.7) adds a threshold over the frequency of negative side effects. Constraint (6.8) denotes the allowed slack on the agent's primary objective, obtained by ignoring negative side effects. The following results show the correctness of constraints (6.6), (6.7).

6.2 Occupancy Measure

Definition 6. *Let $y(c;\pi)$ be the total expected number of times negative side effect c is encountered as per policy π:*

$$y(c;\pi) = \sum_{t=0}^{\infty} \gamma^t P(E_{t+1} = c;\pi) \tag{6.9}$$

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

$$y(c;\pi) = \sum_{u,s,a} y(u,s,a;\pi) \sum_{s'} P(u' = u_\perp, s'|u,s,a) \times \\ \omega(c|u,u_\perp,\sigma = L(s,a,s')) \tag{6.10}$$

Proof. As an output symbol can only be obtained if the NSE controller is in state $u_\perp$, and the starting NSE controller state is $u_{\mathfrak{s}}$, any output symbol can only be obtained from time step 1 onward.

$$\begin{aligned}
y(c;\pi) &= \sum_{t=0}^{\infty} \gamma^t P(E_{t+1} = c;\pi) \\
&= \sum_{t=0}^{\infty} \gamma^t \sum_{u,s,a,u',s'} P((u_t,s_t,a_t,u_{t+1},s_{t+1}) = (u,s,a,u',s'), E_{t+1} = c;\pi) \\
&= \sum_{t=0}^{\infty} \gamma^t \sum_{u,s,a} P((u_t,s_t,a_t) = (u,s,a)) \sum_{u',s'} P(u',s',c|u,s,a) \\
&= \sum_{t=0}^{\infty} \gamma^t \sum_{u,s,a} P((u_t,s_t,a_t) = (u,s,a)) \sum_{s'} P(u_\perp,s',c|u,s,a) \\
&= \sum_{t=0}^{\infty} \gamma^t \sum_{u,s,a} P((u_t,s_t,a_t) = (u,s,a)) \sum_{s'} P(u_\perp,s'|u,s,a) P(c|u,u_\perp,s,a,s') \\
&= \sum_{t=0}^{\infty} \gamma^t \sum_{u,s,a} P((u_t,s_t,a_t) = (u,s,a)) \sum_{s'} P(u_\perp,s'|u,s,a) \omega(c|u,u_\perp,\sigma = L(s,a,s'))
\end{aligned}$$

We can simplify as follows:

$$\sum_{t=0}^{\infty} \gamma^t P((u_t,s_t,a_t) = (u,s,a)) = y(u,s,a;\pi) \tag{6.11}$$

$$y(c;\pi) = \sum_{u,s,a} y(u,s,a;\pi) \sum_{s'} P(u_\perp,s'|u,s,a) \omega(c|u,u_\perp,\sigma = L(s,a,s')) \tag{6.12}$$

□

As a result of this proposition, constraint (6.6), (6.7) model the constraints (3.6) in our original problem, and constraint (6.8) models the constraint (3.5). Thus, the program in Table 6.1 approximately solves the problem (6.6) (up to the accuracy afforded by the learned controller). We empirically run multiple simulations to estimate $y^\star(c)$ to test if the final policy avoids negative side effects.

6.3 CASP Algorithm

Function `Agent(`*Controller, Env*`)`**:**

$obj = \min_x \sum_{u,s,a} y(u,s,a)c(s,a)$

$C = [\sum_a y(u',s',a) = b_0(u',s') + \gamma \sum_{u,s,a} P(u',s'|u,s,a)y(u,s,a)] \quad \forall (u',s')$

$C+ = [y(u,s,a) \geq 0 \quad \forall (u,s,a)]$

$C+ = [y(c) = \sum_{u,s,a} y(u,s,a) \sum_{s'} P(u' = u_\perp, s'|u,s,a) \times \omega(c|u,u_\perp,\sigma) \;\; \forall c \in E]$

$C+ = [y(c) \leq \alpha_c \;\; \forall c \in E]$

$C+ = [\sum_{u,s,a} y(u,s,a)c(s,a) - V^{\pi_b}(s_0) \leq \delta]$

$y^* = LP(obj, C)$

$\pi^\star(a|u,s) = \frac{y^\star(u,s,a)}{\sum_{a'} y^\star(u,s,a')}$

return π^*

The agent takes as input a learned NSE controller and the environment, which is an MDP.

6.4 Experiments

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

6.4.1 Baselines

We compare the performance of our approach with three baselines:

1. executing the *Initial* policy that optimizes the primary objective, with no negative side effects learning involved;

2. a multi-objective approach to mitigate negative side effects (*LMDP*) [26] with a perfect model of negative side effects (*LMDP Optimal*);

3. LMDP with a predictive model of negative side effects learned using approval feedback (*LMDP Learned*).

Since the LMDP can handle only Markov negative side effects, we calculate the non-Markovian negative side effects 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 negative side effect values by performing $10,000$ simulations (e.g., *average negative side effect*=0.5 implies 50% of 10K simulations encountered negative side effect). The experiments are run on two domains: Boxpushing and Navigation (discussed in sections 5.6.1 and 5.6.2).

6.4.2 Slack Utilization

Consider two simple boxpushing instances 4×1 and 4×2, where the shaded area denotes the rug, B denotes the box location, S and G denote start and goal location respectively (Figure 6.1).

(a) Instance (4×1) (b) Instance (4×2)

Figure 6.1: Simple boxpushing instances with Markovian negative side effects.

We evaluate with Markovian negative side effects, slack $\zeta = 5$, and negative side effects threshold $\alpha = 0$. The *Initial* policy always produces negative side effects. To avoid the negative side effects, the agent can wrap the box, incurring an additional cost $+5$, which matches the allowed slack. While our approach with a learned NSE controller avoids the negative side effect by wrapping the box, *LMDP* cannot avoid the negative side effect even with a perfect negative side effects 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 negative side effect.

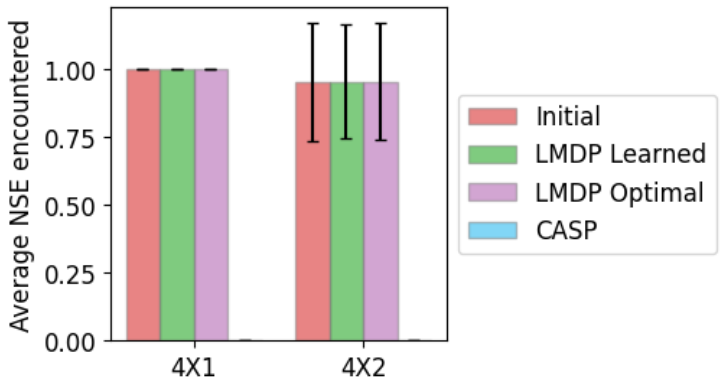


Figure 6.2: Average Markovian negative side effects with standard deviation for simple boxpushing instances, with $\zeta = 5$, demonstrate the LMDP approach's limitation in mitigating Markovian negative side effects due to its slack distribution; CASP has zero negative side effects.

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 [18]. In our setting, the agent is unable to avoid the negative side effect as the wrap action violates the slack allotted to any one 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 negative side effects when feasible.

6.4.3 Results for Markovian NSE

All the Markovian negative side effects are assumed to have the same severity. Results in Figure 6.2, Table 6.2 and Table 6.3 show the average Markovian negative side effects along with standard deviation. The threshold for the negative side effects is set as 0 (i.e., $\alpha = 0$), and the slack is varied as a percentage(%) of the primary objective value (obtained from the initial policy).

Markovian negative side effects are relatively easier to predict and can be avoided with a smaller controller size. We use NSE controllers with 4 nodes to avoid the Markovian negative side effects for both the boxpushing and navigation domains.

The LMDP approaches cannot mitigate the Markovian negative side effects irrespective of the slack provided. However, our approach CASP can mitigate the negative side effects.

Domain	Approach	Slack	Average NSE
Boxpushing	Initial Policy	-	0.97 ± 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

Table 6.2: Effect of varying slack on *Markovian* NSEs in the boxpushing domain with $\alpha = 0$.

Domain	Approach	Slack	Average NSE
Navigation	Initial Policy	-	1.00 ± 0
	LMDP Learned	15%	1.00 ± 0
		20%	1.00 ± 0
		25%	1.00 ± 0
	LMDP Optimal	15%	1.00 ± 0
		20%	1.00 ± 0
		25%	1.00 ± 0
	CASP (#Nodes: 4)	15%	0.0005 ± 0.0223
		20%	0.0005 ± 0.0223
		25%	0.0005 ± 0.0223

Table 6.3: Effect of varying slack on *Markovian* NSEs in the navigation domain with $\alpha = 0$.

For the boxpushing domain, the CASP approach did not find a solution with 15% slack but could avoid negative side effects when the slack increased. The occurrence of negative side effects is reduced to 0 when the slack is set $\geq 20\%$. In the navigation domain, the agent can reduce the occurrence of negative side effects to ~ 0.0005.

6.4.4 Results for Non-Markovian NSE

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

Tables 6.4 and 6.5 show the results on non-Markovian negative side effects for both domains. The threshold for both mild and severe negative side effects is set as 0 (i.e., $\alpha = 0$) and varying slack. Controller sizes were selected based on the accuracies in Table 5.1.

For the boxpushing domain, the CASP approach did not find a solution with 15% slack but could avoid the negative side effects when the slack increased. *LMDP* could not avoid the negative side effects, despite increasing the slack. This shows that besides its limitation in effectively using the slack, *LMDP* cannot mitigate non-Markovian negative side effects.

For the navigation domain, the CASP approach was able to avoid all the negative side effects (mild, severe) for all the three slack values we tested.

6.4.5 Varying threshold and controller sizes

We also test the effect of varying thresholds and controller sizes on the performance. Figures 6.3 and 6.4 shows the performance with slack $\zeta = 15\%$ on both domains, as we vary the controller size and thresholds α_i for mild and severe negative side effects.

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

We test our approach for three different threshold configurations:

1. Threshold for severe NSE is 0.1, and for mild NSE is 0.1
2. Threshold for severe NSE is 0.1, and for mild NSE is 0.2
3. Threshold for severe NSE is 0.3, and for mild NSE is 0.3

In the boxpushing domain, our approach with four is not able to perform well for the third threshold configurations, and our approach with six nodes is not able to perform well for the second and third threshold configurations. Results with eight nodes achieve the best performance overall.

In the navigation domain, our approach with five nodes was not able to find a solution for the first two threshold configurations, and therefore we report the initial policy value. Results with seven nodes achieve the best performance overall. The performance worsens with nine nodes on the navigation problem since the model overfits the training data.

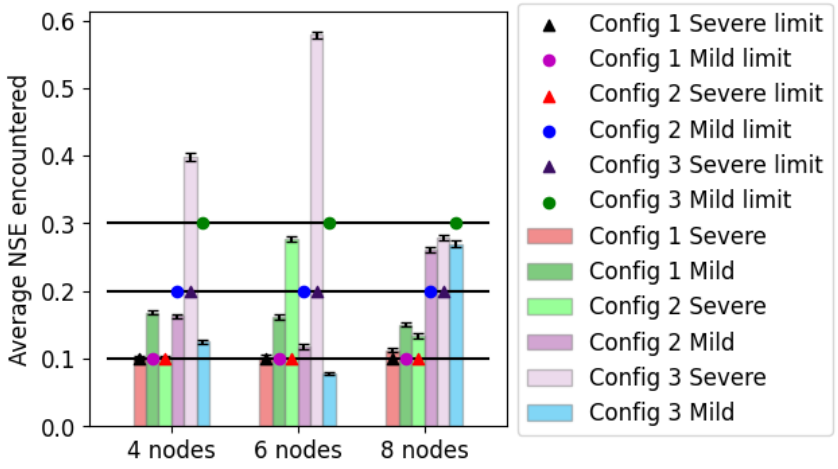


Figure 6.3: Results with varying controller size and NSE threshold for the boxpushing domain, when $\zeta = 15\%$, along with standard error. Configuration 1: (0.1 severe 0.1 mild), Configuration 2: (0.1 severe 0.2 mild), Configuration 3: (0.2 severe 0.3 mild).

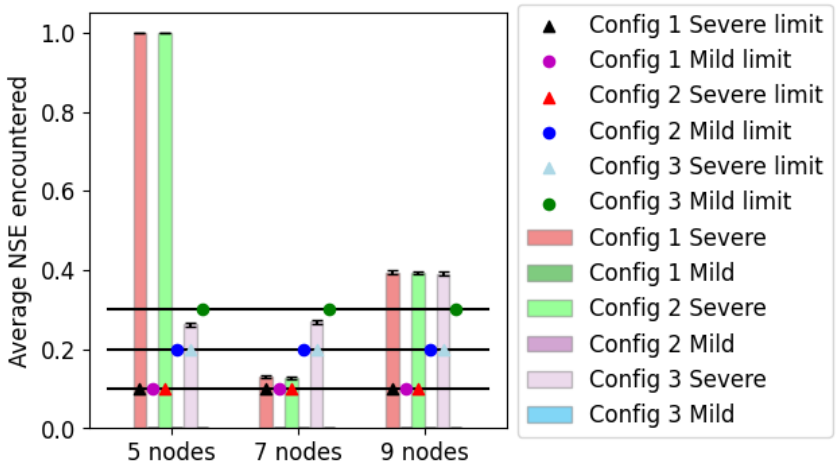


Figure 6.4: Results with varying controller size and NSE threshold for the navigation domain, when $\zeta = 15\%$, along with standard error. Configuration 1: (0.1 severe 0.1 mild), Configuration 2: (0.1 severe 0.2 mild), Configuration 3: (0.2 severe 0.3 mild).

Chapter 7

Conclusions and Future Work

In this thesis, we primarily discuss the problem of encountering negative side effects when operating in open-world environments due to policies learned based on incomplete models of the environment. Our primary contributions are: (1) designing a model based on Finite State Controllers (FSCs) that can predict the severity of negative side effects for a given trajectory; (2) learning the model parameters using observed data containing state-action trajectories and the severity of the associated negative side effect.; (3) developing a constrained MDP model that uses information from the underlying MDP and the learned model for planning while avoiding negative side effects; and (4) evaluating the performance of our approach on two domains, Boxpushing and Navigation.

7.1 Conclusions

We present CASP, a paradigm to learn and mitigate Markovian and non-Markovian NSEs, with bounded-performance guarantees with respect to the primary objective value and NSE occurrence.

7.1.1 Learning NSE Controller

We represent the negative side effects using an NSE controller modeled as a finite state controller. The training data for our results are generated using an ϵ-greedy version of the initial policy using the optimal primary value function V^*. We model the optimization problem for learning the NSE controller parameters using the maximum likelihood estimation(MLE) and Expectation-Maximization(EM) algorithm. We further showed how to compute the expected sufficient statistics using a forward-backward algorithm and use the KKT conditions [5] to solve the optimization problem analytically to obtain improved estimates of δ, and ω parameters. Our results with varying controller sizes show the effectiveness of our approach

in learning an NSE controller with high accuracy. We are able to learn an NSE controller with more than 90% accuracy for both domains.

7.1.2 CASP: Controller-Assisted Safe Planning

The problem of mitigating negative side effects is formulated as a constrained MDP, and a side effects-minimizing policy is computed by integrating the NSE controller with our planning model (CASP). Our approach efficiently utilizes the allowed slack in states where it is most needed instead of being evenly distributed across all states. Our results with varying slack, controller sizes, and negative side effect thresholds on Markovian and non-Markovian negative side effects demonstrate the effectiveness of our approach in mitigating negative side effects. Our approach is able to avoid NSEs Markovian and non-Markovian NSEs in both domains when the threshold for them is set as 0.

7.2 Future Work

Our current approach is limited to domains that can be represented as MDPs. In the future, one may aim to extend our technique to partially observable settings. We currently only consider discrete settings. Extending our approach to handle continuous state space is another interesting direction for future research. We can also look at better handling noise in the training data and explore a different method for representing negative side effects.

Related Publications

Aishwarya Srivastava, Sandhya Saisubramanian, Praveen Paruchuri, Akshat Kumar, and Shlomo Zilberstein, *Planning and Learning for Non-markovian Negative Side Effects Using Finite State Controllers*. (Association for the Advancement of Artificial Intelligence - AAAI 2023, accepted as a full paper for oral presentation.)

Link to public implementation: `https://github.com/SriAish/NSE_PlanningAgent`

Bibliography

[1] P. Alizadeh Alamdari, T. Q. Klassen, R. Toro Icarte, and S. A. McIlraith. Be considerate: Avoiding negative side effects in reinforcement learning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pages 18–26, 2022.

[2] E. Altman. *Constrained Markov Decision Processes*. 2021.

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

[4] C. C. Bennett and K. K. Hauser. Artificial intelligence framework for simulating clinical decision-making: A markov decision process approach. *Artificial intelligence in medicine*, 57 1:9–19, 2013.

[5] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

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

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

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

[9] J. T. Folsom-Kovarik, G. R. Sukthankar, and S. L. Schatz. Tractable pomdp representations for intelligent tutoring systems. *ACM Trans. Intell. Syst. Technol.*, 4:29:1–29:22, 2013.

[10] D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. Dragan. Inverse reward design. In *Advances in Neural Information Processing Systems*, pages 6765–6774, 2017.

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

[12] B. Hibbard. Avoiding unintended AI behaviors. In *International Conference on Artificial General Intelligence*, pages 107–116. Springer, 2012.

[13] V. Krakovna, L. Orseau, M. Martic, and S. Legg. Penalizing side effects using stepwise relative reachability. In *AI Safety Workshop, IJCAI*, 2019.

[14] V. Krakovna, L. Orseau, R. Ngo, M. Martic, and S. Legg. Avoiding side effects by considering future tasks. In *Advances in Neural Information Processing Systems*, 2020.

[15] H. Lakkaraju, E. Kamar, R. Caruana, and E. Horvitz. Identifying unknown unknowns in the open world: Representations and policies for guided exploration. In *AAAI Conference on Artificial Intelligence*, 2016.

[16] N. Meuleau, K. Kim, L. P. Kaelbling, and A. R. Cassandra. Solving pomdps by searching the space of finite policies. In K. B. Laskey and H. Prade, editors, *International Conference on Uncertainty Artificial Intelligence*, pages 417–426, 1999.

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

[18] L. E. Pineda, K. H. Wray, and S. Zilberstein. Revisiting multi-objective MDPs with relaxed lexicographic preferences. In *Proceedings of the AAAI Fall Symposium Series*, 2015.

[19] M. L. Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[20] R. Ramakrishnan, E. Kamar, D. Dey, E. Horvitz, and J. Shah. Blind spot detection for safe sim-to-real transfer. *Journal of Artificial Intelligence Research*, 67:191–234, 2020.

[21] R. Ramakrishnan, E. Kamar, B. Nushi, D. Dey, J. A. Shah, and E. Horvitz. Overcoming blind spots in the real world: Leveraging complementary abilities for joint execution. In *AAAI Conference on Artificial Intelligence*, 2019.

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

[23] S. Russell. Provably beneficial artificial intelligence. *27th International Conference on Intelligent User Interfaces*, 2022.

[24] S. J. Russell, D. Dewey, and M. Tegmark. Research priorities for robust and beneficial artificial intelligence. *AI Mag.*, 36:105–114, 2015.

[25] S. Saisubramanian, E. Kamar, and S. Zilberstein. Mitigating the negative side effects of reasoning with imperfect models: A multi-objective approach. In *Adaptive Agents and Multi-Agent Systems*, 2020.

[26] S. Saisubramanian, E. Kamar, and S. Zilberstein. A multi-objective approach to mitigate negative side effects. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, pages 354–361, 2020.

[27] S. Saisubramanian, E. Kamar, and S. Zilberstein. Avoiding negative side effects of autonomous systems in the open world. *Journal of Artificial Intelligence Research*, 74:143–177, 2022.

[28] S. Saisubramanian, S. C. Roberts, and S. Zilberstein. Understanding user attitudes towards negative side effects of AI systems. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pages 368:1–368:6, 2021.

[29] S. Saisubramanian and S. Zilberstein. Mitigating negative side effects via environment shaping. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1640–1642, 2021.

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

[31] S. Saria and A. Subbaswamy. Tutorial: Safe and reliable machine learning. *ArXiv*, abs/1904.07204, 2019.

[32] R. Shah, D. Krasheninnikov, J. Alexander, P. Abbeel, and A. D. Dragan. Preferences implicit in the state of the world. *ArXiv*, abs/1902.04198, 2018.

[33] I. Sim. Mobile devices and health. *The New England journal of medicine*, 381 10:956–968, 2019.

[34] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 16:285–286, 2005.

[35] P. S. Thomas, B. C. da Silva, A. G. Barto, S. Giguere, Y. Brun, and E. Brunskill. Preventing undesirable behavior of intelligent machines. *Science*, 366:1004 – 999, 2019.

[36] A. M. Turner, D. Hadfield-Menell, and P. Tadepalli. Conservative agency via attainable utility preservation. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 385–391, 2020.

[37] Wikipedia contributors. Finite-state machine — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Finite-state_machine&oldid=1135387389`, 2023. [Online; accessed 19-February-2023].

[38] K. H. Wray, S. Zilberstein, and A.-I. Mouaddib. Multi-objective MDPs with conditional lexicographic reward preferences. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence*, pages 3418–3424, 2015.

[39] S. Zhang, E. H. Durfee, and S. Singh. Querying to find a safe policy under uncertain safety constraints in Markov decision processes. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, pages 2552–2559, 2020.

[40] S. Zhang, E. H. Durfee, and S. P. Singh. Minimax-regret querying on side effects for safe optimality in factored Markov decision processes. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 4867–4873, 2018.

[41] S. Zilberstein. Building strong semi-autonomous systems. In *AAAI Conference on Artificial Intelligence*, 2015.