Stochastic Safe Action Model Learning

Zihao Deng LEOLEVEL104@GMAIL.COM
*Amazon**

Brendan Juba BJUBA@WUSTL.EDU
Washington University in St. Louis

Editors: Steve Hanneke and Tor Lattimore

Abstract

Hand-crafting models of interactive domains is challenging, especially when the dynamics of the domain are stochastic. We thus wish to automatically learn such models instead. In this work, we propose an algorithm to learn stochastic planning models where the distribution over the sets of effects for each action has a small support, but the sets may set values to an arbitrary number of attributes. This class captures many benchmark domains, in contrast to prior work that assumed independence of the effects on individual attributes. Our algorithm has polynomial time and sample complexity when the support size is bounded by a constant. Importantly, our method is safe in that we learn offline from example trajectories and we guarantee that actions are only permitted in states where our model of the dynamics is accurate. Moreover, we guarantee approximate completeness of the model, in the sense that if the examples are achieving goals from some distribution, then with high probability there will exist plans in our learned model that achieve goals from the same distribution.

Keywords: offline learning; stochastic planning; method of moments

1. Introduction

In classical high-level task planning, a domain model describes the interaction between an environment and the planning agent. The domain model is usually specified in a formal language and includes an action model, which specifies which actions can be in a plan and how they work. For stochastic environments, such formal languages include PPDDL (Younes and Littman, 2004) and RDDL (Sanner, 2010), for example. The action model describes the effects of the actions on the environment's state, and the preconditions that must be true in order for the action to be taken. Creating a planning domain model and action model, however, is a notoriously hard knowledge-engineering task, especially when the domain is stochastic. Many approaches have thus been proposed to automatically learn the domain model (Pasula et al., 2007; Mourão et al., 2012; Zhuo and Kambhampati, 2013; Martínez et al., 2015, 2016; Mao et al., 2022; Juba and Stern, 2022; Lamanna and Serafini, 2024). This learning problem is more difficult than the problem for deterministic domains. Indeed, in a deterministic environment, we can learn that some state attributes are not part of the effect as long as they are unchanged after a transition, and that they are part of an effect if they change. However, when the effects are stochastic, it is possible that a change does not appear in the transition and yet has significant probability of occurring. Consequently, many observations are needed: if there are even small errors in our estimates of these probabilities, it can accumulate over the course of execution, leading to wildly inaccurate estimates of the trajectory. Moreover, even in these simple

* Work performed while affiliated with Washington Universtiy in St. Louis

domain models where effects only set some attributes to take specific values, we may not observe whether or not an effect occurred if the attribute already had that value in the pre-state. Therefore, in any given transition, we generally do not know which attributes would have been set by the random effects of the action, even when the states are fully observed.

Our goal is to safely learn a stochastic action model that is guaranteed to be accurate and complete. For safety, we use *offline learning* of the action model, assuming that demonstrations of competent performance of the domain have been provided e.g., by a human controller, and we seek a guarantee that the model is sufficiently accurate. This is similar to offline reinforcement learning (Levine et al., 2020; Jiang and Xie, 2025), except that we wish to learn a reusable model that allows solving many goals, and we learn high-level task planning representations (with concise representations in PPDDL); we defer a more detailed comparison to Section 2.1.1. For such safety, we produce a conservative model that only permits actions to be taken when the learned model can be guaranteed to accurately describe the actual distribution of transitions in the environment. We note that such a guarantee is sufficient for us to ensure safety via constraints on the policy during planning: if the model predicts we never enter an "unsafe" state and the model is correct, then indeed the agent remains in "safe" states, avoiding potential danger. For the model to be useful, we must further ensure that this conservative model is permissive enough to allow successful performance in the domain.

Previous work that tackles safe learning of high-level task planning models of stochastic environments (Juba and Stern, 2022) achieved safety and completeness under an assumption that the effects of actions on each attribute are *independent* random variables (a.k.a. *"factored"*). Here, we relax that assumption and provide an algorithm for stochastic environments with dependent effects. Instead, we have an assumption that each action has a small number of *sets* of effects. For example, consider a noisy blocks world domain where there is a pickup action that has three possible outcomes: either the block is picked up off the table and in hand, e.g. with 75% chance; or there is no effect with 20% chance; or *all* of the blocks are on the table with nothing on top of them (e.g., because the arm bumped the table), with 5% chance. These are three distinct sets of effects; in the last outcome, there are many effects since the action has an effect on every block, and these are all correlated. Our algorithm will be able to handle this domain, but the prior work of Juba and Stern cannot. Or, as a second example, suppose we wish to represent the position of an object with Boolean attributes, where we use a distinct attribute to represent the presence or absence of the object at that position. To achieve this, effects must update the position-encoding attributes in a correlated way; if the position attributes were updated independently (as in the environments that Juba and Stern capture) then either the action's effect is deterministic, or else there is nonzero probability of the object moving to zero or two (or more) positions following a transition. So such environments cannot be represented with independent effects across attributes. Again, our method can learn such environments but Juba and Stern's algorithm cannot.

The prior work by Juba and Stern (2022) gave an efficient and safe method for learning the actions' preconditions that carries over to our setting, so we focus on learning the distribution of effects for each action. We use the method of moments pioneered by Pearson (1936) (Wooldridge, 2001; Hall, 2004; Ney, 1985; Gibson, 2021; Newey and West, 1987; Ogaki, 1993; Mátyás et al., 1999, for example) to learn the distribution of effects. In particular, work by Anandkumar et al. (2014) demonstrated that the method of moments can efficiently fit a wide class of probabilistic models, given a certain "identifiability" assumption, that the moments uniquely determine the parameters. When the parameters are "generic" numbers (in the sense of algebraic geometry), identifiability is

guaranteed; unfortunately, our discrete effect vectors are not "generic." Worse, in our setting, the full effects distribution actually may not be identifiable.

Contributions. In this work, we *first* show that for the kinds of simple discrete effects used in classical planning models (e.g., PPDDL environments in the International Planning Competition probabilistic planning track), a moderate number of moments suffices to ensure identifiability of fully observed distributions (Sec. 2.3). *Second*, we identify this family of environments as one where efficient learning of models is tractable. *Third,* we observe that for an accurate model of the domain, since the "unobserved" effects are those that leave the fluents with the same values, it is sufficient for the observed marginals to be consistent—it is not necessary to solve the more demanding problem of predicting the missing entries. *Fourth,* we give a polynomial-time algorithm to construct a set of stochastic effects that is consistent with the observed marginals (under conditions where imputing the missing entries may be impossible). *Finally,* we show that we can add additional preconditions that ensure that the domain model only permits actions to be taken when the effect distribution in our learned model is guaranteed to be close to the true distribution; in particular, we show that these additional preconditions do not significantly reduce the completeness of the model, relative to the distribution on example trajectories. We thus obtain a sound and approximately complete model.

Related work. Learning planning models for stochastic domains has been considered previously. Pasula et al. (2007) formulated the learning problem as optimizing an objective that could not be tractably optimized. Unfortunately, the greedy heuristic they proposed cannot provide the soundness or approximate completeness guarantees that we seek. Martínez et al. (2015, 2016) proposed a method that could handle exogeneous transitions, but with essentially similar limitations. On the other hand, Mourão et al. (2012) and Lamanna and Serafini (2024) assume that the actual domain is deterministic, and merely the observations are corrupted by stochastic noise. (They also do not provide the kind of guarantees that we seek.) Mao et al. (2022) proposed to use a neural network to predict missing parts of an action model. This again voids any expectation of safety and prevents the representation from being used in standard planners.

Our algorithm for constructing a consistent effect distribution solves a problem resembling low-rank matrix completion (e.g., Candès and Recht (2012)) for tensors with a nonuniform observation model. Prima facie, it is impossible since the solution may not be identifiable. As discussed above, it crucially suffices for us to only match the distribution's marginals for which we have observations. The "completed" entries may be wrong, but the additional preconditions ensure that the model only permits policies to take actions where the distribution of their effects is guaranteed to be modeled correctly. The upshot is that it would not be possible for us to use an "off-the-shelf" algorithm for low-rank tensor completion for our problem such as Yang et al. (2021), because the lack of identifiability in general violates the assumptions required for those algorithms.

2. Preliminaries

We now recall our setting and the Stochastic Safe Action Model Learning (SAM) problem. Subsequently, we introduce groundwork for the method of moments.

2.1. Stochastic Safe Action Model Learning

We formulate the problem in terms of grounded PPDDL representations for simplicity, but our method may be extended to lifted representations following Juba et al. (2021). We also do not

consider conditional effects or action costs. A domain $D = \langle F, A, M \rangle$ consists of a set F of *fluents*, a set A of *actions*, and an *action model* M for these actions, each described in more detail below.

A fluent f is a variable representing a fact that may or may not hold in the environment at some point in time. A state $\boldsymbol{s}$ is a vector of assignments of Boolean values s_f to each respective fluent f in F. We will abuse notation by denoting the indicator functions $1[s_f = 1]$ as f and $1[s_f = 0]$ as $\neg f$, which we refer to as *literals*. We will also sometimes write $s_{\neg f}$, which means $\neg s_f$. An action is an operation that can be taken by the planning agent to change the values in $\boldsymbol{s}$, hence transitioning to another state $\boldsymbol{s'}$. It is given by a *name* (and, in the case of lifted models, a set of *arguments*). The action model M defines *preconditions* $pre_M(a)$ and *effects* $eff_M(a)$ for each action $a \in A$. A precondition of an action is a Boolean formula on the fluents, with the interpretation that the precondition must be satisfied on the current state $\boldsymbol{s}$ to allow the action a to be taken. We say the action is then *legal*. In the domains we consider, these preconditions will be conjunctions, i.e., an AND of literals, but we will produce action models in which the preconditions are conjunctive normal form formulas: an AND of ORs of literals.

For the effects, we consider a fragment of PPDDL in which the effects consist of a single "`probabilistic`" block for each action: i.e., for each action, there is a sequence of r partial assignments $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_r$ where each $\boldsymbol{e}_i$ is associated with a respective probability p_i. That is, when action a is taken, with probability p_i, the next state $\boldsymbol{s'}$ agrees with the partial assignment $\boldsymbol{e_i}$, and any fluents not set in $\boldsymbol{e_i}$ remain the same value as in the previous state $\boldsymbol{s}$. We assume that the number of effects $r = r(a)$ for each a is at most a small constant, which is consistent with PPDDL benchmarks in the IPC probabilistic tracks, where generally $r \leq 5$.[1] This fragment was first considered in the context of stochastic planning by Blum and Langford (2000), and was the representation considered for learning by Pasula et al. (2007). The action model thus specifies the probability of transitioning from a state $\boldsymbol{s}$ to another state $\boldsymbol{s'}$ by applying a, denoted by $Pr_M[\boldsymbol{s'}|a, \boldsymbol{s}]$.

A PPDDL planning problem $\Pi = \langle D, \boldsymbol{s}_I, \boldsymbol{s}_G \rangle$ consists of a domain D, a starting state $\boldsymbol{s}_I$, and a goal partial assignment $\boldsymbol{s}_G$. A solution to a PPDDL planning problem is a policy π, which is a mapping from the states to the actions $\pi : \{0,1\}^F \rightarrow A$. To execute a policy π on the planning problem Π is to repeatedly apply actions according to the policy π given the current state, starting by applying action $\pi(\boldsymbol{s}_I)$ at the starting state $\boldsymbol{s}_I$. The execution ends when a state consistent with $\boldsymbol{s}_G$ is reached, or some other condition is satisfied, such as a time-out number of actions taken, after which the agent gives up. A trajectory $\mathcal{T}$ is a an alternating sequence of states and actions of the form $\langle \boldsymbol{s}_0, a_0, \boldsymbol{s}_1, a_1, \ldots, a_{|\mathcal{T}|}, \boldsymbol{s}_{|\mathcal{T}|} \rangle$. We assume the trajectory includes the values of each fluent at each state, and an identifier of which action was taken for each transition. Each execution of a policy π creates a trajectory starting from $\boldsymbol{s}_0 = \boldsymbol{s}_I$. The length $|\mathcal{T}|$ of trajectory $\mathcal{T}$ is the number of actions taken.

In the SAM Learning problem, there is an arbitrary probability distribution on problems in a fixed domain D and policies for D. Trajectories $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_m\}$ are sampled by first drawing an independent ith problem $\langle D, \boldsymbol{s}_I^{(i)}, \boldsymbol{s}_G^{(i)} \rangle$ and policy $\pi^{(i)}$ from the distribution, and then executing $\pi^{(i)}$ in D from $\boldsymbol{s}_I^{(i)}$ to produce $\mathcal{T}_i$. Given this sample of trajectories (implicitly with identifiers for the names of fluents F and actions A) as input, we produce as output an action model $\hat{M}$, thus giving a

1. We do not consider learning the *conditional* effects are sometimes used in these planning benchmarks. Learning an action model with safety guarantees in the presence of conditional effects is much more challenging, even in a deterministic environment (Mordoch et al., 2024).

learned domain representation $\hat{D} = \langle F, A, \hat{M} \rangle$. For a given ϵ and δ, we require that with probability $1 - \delta$, we obtain $\hat{M}$ such that

1. *Safety* For any policy π that takes L legal actions in $\hat{D}$, the distribution on trajectories obtained by π in $\hat{D}$ is ϵ-close in total variation distance to the distribution obtained by π in D, and the actions of π are legal in D as well.
2. *Approximate completeness* For $\langle D, \boldsymbol{s'}_I, \boldsymbol{s'}_G \rangle$ and π' sampled independently from the distribution, with probability $1 - \epsilon$ there is a policy $\hat{\pi}$ that takes legal actions in $\hat{D}$ and reaches $\boldsymbol{s'}_G$ from $\boldsymbol{s'}_I$ in $\hat{D}$ with probability within ϵ of the probability that π' reaches $\boldsymbol{s'}_G$ from $\boldsymbol{s'}_I$ in D.

In addition to the running time of an algorithm for this problem, we will show that there is a polynomial bound on the number of trajectories $m = m(|F|, |A|, L, \epsilon, \delta)$ needed to obtain such an action model with probability $1 - \delta$. This is the *sample complexity* of the problem.

2.1.1. COMPARISON TO OFFLINE REINFORCEMENT LEARNING

With the formal setting in place, we can give a more nuanced comparison of our problem setting and guarantees to offline reinforcement learning. (We refer the reader to the survey of Jiang and Xie (2025) for the terminology of offline reinforcement learning.) Our task is a kind of model-based learning, where the main model restriction is that the support of the transition distribution (conditioned on the current state and action) is assumed to be bounded by a small value r. As we'll see, this means that the matrix of moments of the changes in values of state attributes has rank r. This *does not* mean that the state transition matrix is low rank (i.e., a low-rank MDP). Unlike typical reinforcement learning settings, these outcomes simply set a subset of the state attributes to a fixed assignment (one of the r partial assignments at random). In particular, we could have actions that leave the state unchanged, thus yielding a transition matrix of rank equal to the size of the state space. There is in general no low-dimensional summary of our states.

We indeed work in an offline setting, and the techniques used to obtain our "safety" guarantee are similar to the pessimism principle used in offline reinforcement learning: we use confidence intervals on the transition probabilities to obtain a version space of candidate models, and use additional "preconditions" to forbid actions when these transitions cannot be accurately estimated. On the other hand, the completeness guarantees we obtain are similar to single-policy coverage: we guarantee that planning with the learned model yields a success rate that matches that of the sampling policy. As in classical task planning, however, we focus strictly on reachability goals rather than total reward. But, this formulation has the benefit that new reachability goals can be defined on the state space and we guarantee generalization to reachability goals from the sampling distribution. By contrast, reinforcement learning usually produces policies tied to a specific fixed reward function. There is work on *goal-conditioned* reinforcement learning that is not subject to this restriction (Ghosh et al., 2023; Park et al., 2023; Wang et al., 2024; Ke et al., 2025; Zhang et al., 2025; Myers et al., 2026), but these goal-conditioned policies must solve the associated planning problem which is generally extremely hard (Littman et al., 1998), and we still do not have good structured families of such goal-conditioned policies. Thus, we generally cannot get polynomial-time guarantees for rich environments with goal-conditioned reinforcement learning.

2.2. Tensor decomposition

A degree-d, dimension-n tensor T is an array of numbers $T[i_1, \ldots, i_d]$ indexed by d indices, where each index $i_j \in [n]$ for $j \in [d]$. For example, a matrix is a degree-2 tensor. To decompose a tensor

T is to represent each of its element as the weighted sum of r products:

$$T[i_1, \ldots, i_d] = \sum_{k=1}^{r} w_k v_k^{(1)}[i_1] \cdots v_k^{(d)}[i_d],$$

for vectors $\boldsymbol{v}_k^{(j)}$ of dimension n. Equivalently, we can write it as: $T = \sum_{k=1}^{r} w_k \boldsymbol{v}_k^{(1)} \otimes \cdots \otimes \boldsymbol{v}_k^{(d)}$, where $\otimes$ is the outer product. The minimum number of terms r in such possible decompositions is called the tensor rank of T. We refer to the set of vectors $V^{(j)} = [\boldsymbol{v}_1^{(j)}, \ldots, \boldsymbol{v}_r^{(j)}]$ as the jth mode of this tensor decomposition, where $j \in [d]$. If all modes $V^{(j)}$ are the same $V = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r]$, we can write the decomposition as $T = \sum_{k=1}^{r} w_k \boldsymbol{v}_k^{\otimes d}$, where $\boldsymbol{v}_k^{\otimes d}$ is the outer product of a vector $\boldsymbol{v}_k$ with itself d times. Note that for positive w_k (or odd d), by rescaling each vector $\boldsymbol{v}_k$ by $w_k^{1/d}$, we can obtain the same tensor while dropping w_k. We know that in general, matrix decompositions are not unique. By contrast, the decomposition for tensors is often unique. In order to establish that the tensor decomposition is unique, we leverage Kruskal's theorem (Kruskal, 1977). It is the cornerstone of establishing identifiability in many settings (Gu, 2025; Allman and Rhodes, 2008; Fang et al., 2019; Culpepper, 2019; Chen et al., 2020; Fang et al., 2021; Xu, 2017).

Theorem *(Kruskal, 1977) Suppose that a degree-3 tensor T has a decomposition $\sum_{k=1}^{r} \boldsymbol{a}_k \otimes \boldsymbol{b}_k \otimes \boldsymbol{c}_k$. Let $A = [\boldsymbol{a}_1, \ldots, \boldsymbol{a}_r]$, $B = [\boldsymbol{b}_1, \ldots, \boldsymbol{b}_r]$, and $C = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_r]$ denote matrices with these vectors as columns. Suppose every set of I columns of A are linearly independent, every set of J columns of B are linearly independent, and every set of K columns of C are linearly independent. If $I + J + K \geq 2r + 2$, then this tensor decomposition with r components is unique up to permutation.*

2.3. Unique Boolean Decomposition

We now show that for any tensor power of Boolean components that is logarithmic in the number of components, the tensor decomposition is unique. Thus, for the tensors corresponding to logarithmic moments of a discrete distribution, the tensor decomposition recovers the components. This can be implicitly seen in Chen and Moitra (2019), but for completeness we present it here and include a full proof in the appendix. Suppose that we are given a degree-$2k+1$ tensor $\sum_{i=1}^{r} w_i \boldsymbol{v}_i^{\otimes(2k+1)}$, where $\boldsymbol{v}_i \in \{0,1\}^d$. The goal is to obtain $\boldsymbol{v}_i$. We will re-shape it and obtain a degree-3 tensor $\sum_i^r w_i \boldsymbol{v}_i \otimes flat(\boldsymbol{v}_i^{\otimes k}) \otimes flat(\boldsymbol{v}_i^{\otimes k})$, where $flat(M)$ denotes the tensor rearranged into a vector.

By Kruskal's Theorem, we can argue that the tensor decomposition of $\sum_i^r \boldsymbol{v}_i \otimes flat(\boldsymbol{v}_i^{\otimes k}) \otimes flat(\boldsymbol{v}_i^{\otimes k})$ is unique up to permutation. Indeed, suppose $V = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r\}$, and

$$V^{\otimes k} = \{flat(\boldsymbol{v}_1^{\otimes k}), \ldots, flat(\boldsymbol{v}_r^{\otimes k})\}.$$

According to Lemma 1, $V^{\otimes k}$ must be full rank when $k = O(\log r)$ (see Appendix B for the proof):

Lemma 1 *For all $k, n \in \mathbb{N}$ and $S \subseteq \{0,1\}^n$, if $|S| \leq 2^{k+1} - 2$, then $S^{\otimes k}$ is linearly independent.*

So, if V has at least two distinct 0-1 vectors, we have $rank(V) + rank(V^{\otimes k}) + rank(V^{\otimes k}) \geq 2 + 2r$. Due to Kruskal's Theorem, when $k = O(\log r)$, the decomposition is unique. Although the statement of Kruskal's Theorem does not include weights w_i, we can recover these from the scaling: Since $\boldsymbol{v}_i$ is known to be a 0-1 vector, we can still read out the 0-1 information from the zero and nonzero values of the decomposed vectors, and the value of w_i will be the product of the the nonzero values of the three modes for each i. Hence, we can obtain our decomposition by computing the tensor decomposition of this $d \times kd \times kd$ reshaping using any tensor decomposition method.

3. Overview of Our Approach

We now give an overview of our approach to solving the Stochastic SAM Learning problem.

3.1. Learning preconditions

Preconditions can be learned following the same approach described in Juba and Stern (2022): For the class of domains we consider, for each $(\boldsymbol{s}, a, \boldsymbol{s'}) \in T$, where $T \in \mathcal{T}$, if a literal is not in the previous state then it cannot be a precondition. Precisely: $s_{\neg\ell} \Rightarrow \ell \notin pre(a)$ where $pre(a)$ is the preconditions of action a according to the actual action model M^*. We can thus learn the preconditions by initially assuming every action has all the literals as preconditions, and then applying this rule to remove literals from the preconditions as needed. More specifically, let $\mathcal{T}(a)$ be all the $\langle \boldsymbol{s}, a, \boldsymbol{s'} \rangle$ triplets for action a. State $\boldsymbol{s}$ is the pre-state and $\boldsymbol{s'}$ is the post-state of action a. We (initially) set the precondition of a to be the set of all ℓ such that $\forall \langle \boldsymbol{s}, a, \boldsymbol{s'} \rangle \in \mathcal{T}(a)\ s_\ell = 1$.

3.2. Learning effects

We now need to recover the sets of effects $\{\boldsymbol{e}_i\}$ for each action a, where each set of effects $\boldsymbol{e}_i$ has a corresponding probability p_i. These estimated stochastic effects, together with our learned preconditions, will give an approximate action model $\hat{M}$ solving the Stochastic SAM Learning problem. To recover the effects from our observations, we first estimate moments of the effect indicator variables, i.e., where the indicator for literal ℓ is 1 if ℓ was an effect of a and 0 otherwise. The values of these indicators can only be determined from the data if ℓ was not true in the previous state. Since the distribution of effects is assumed to only depend on the action taken (and not the previous state), we can estimate these probabilities by conditioning on ℓ being false in the previous state, i.e., counting the number of such transitions in the example trajectories. The problem that arises is that we may not have such states where a is taken and ℓ is false; we return to this later.

For illustration, let's start with degree-3 moments: Let $n = 4$ be the total number of literals ℓ. Suppose action a has $r = 3$ effects: $\boldsymbol{e}_1 = \{\ell_1, \ell_3\}$, $\boldsymbol{e}_2 = \{\ell_1, \ell_2\}$, and $\boldsymbol{e}_3 = \{\ell_2\}$. the probability values we observed and approximated are $p_{\ell_i,\ell_j,\ell_k} := Pr\left[s'_{\ell_i}, s'_{\ell_j}, s'_{\ell_k} | s_{\neg\ell_i}, s_{\neg\ell_j}, s_{\neg\ell_k}, a\right]$. These degree-3 moment data can be arranged into a 3-dimensional tensor T_a where each entry is the probability p_{ℓ_i,ℓ_j,ℓ_k}, and indexed by three literals ℓ_i, ℓ_j, ℓ_k. Its value will be the *sum of all* p_i *for all effects* $\boldsymbol{e}_i$ *that induces* ℓ_i, ℓ_j, ℓ_k *being true* under the condition that $\neg\ell_i, \neg\ell_j, \neg\ell_k$ are true. We can then extract the effects from these moment data. Indeed, the effect sets we want to recover can be formulated as 0-1 vectors such as $\boldsymbol{e}_1 = (1, 0, 1, 0)$, $\boldsymbol{e}_2 = (1, 1, 0, 0)$, $\boldsymbol{e}_3 = (0, 1, 0, 0)$. The degree-3 moments can arranged into the tensor

$$T_a = \sum_{i=1}^{r} p_i \boldsymbol{e}_i^{\otimes 3} = p_1 \boldsymbol{e}_1^{\otimes 3} + p_2 \boldsymbol{e}_2^{\otimes 3} + p_3 \boldsymbol{e}_3^{\otimes 3}, \tag{1}$$

where p_i are the probabilities of these effects. Since Eq. 1 is a feasible decomposition of T_a, when it is unique, it must be the only decomposition T_a. Thus, if we compute a tensor decomposition of T_a, the vectors $\boldsymbol{e}_1$, $\boldsymbol{e}_2$, and $\boldsymbol{e}_3$ in the decomposition are the effects of the action a, with the associated scalings p_1, p_2, p_3 as the probabilities of the respective effects. As long as the number of effects e_i is small and the components are linearly independent (established in Sec. 2.3), then this problem can be solved in polynomial time by existing algorithms (for example, see Schramm and Steurer

	top	¬top	left	¬left
top	1/2	?	0	0
¬top	?	?	?	?
left	0	?	0	?
¬left	0	?	?	0

(*a*) top slice

	top	¬top	left	¬left
top	?	?	?	?
¬top	?	1/2	0	0
left	?	0	0	?
¬left	?	0	?	0

(*b*) ¬top slice

	top	¬top	left	¬left
top	0	?	0	?
¬top	?	0	0	?
left	0	0	0	?
¬left	?	?	?	?

(*c*) left slice

	top	¬top	left	¬left
top	0	?	?	0
¬top	?	0	?	0
left	?	?	?	?
¬left	0	0	?	0

(*d*) ¬left slice

Figure 1: Third moment tensor for effects of updown action in toy example

(2017) or Anandkumar et al. (2014)). Since the number of entries in this tensor is $O(n^3)$, we can find this decomposition in $poly(n)$ time.

The decomposition is indeed unique, following Sec. 2.3. Depending on the number of outcomes r, we construct degree $d = O(\log r)$ moments, $p_{\ell_{i_1},\ldots,\ell_{i_d}} := Pr[s'_{\ell_{i_1}}, \ldots, s'_{\ell_{i_d}} | s_{\neg\ell_{i_1}}, \ldots, s_{\neg\ell_{i_d}}, a]$ and the tensor decomposition we want to obtain is: $T_a = \sum_{i=1}^{r} p_i \boldsymbol{e}_i^{\otimes d}$, where $\boldsymbol{e}_i \in \{0,1\}^n, \forall i \in [r]$. For such d, following Sec. 2.3, we can reshape this tensor into a degree-3 tensor and use the existing tensor decomposition algorithms to obtain the unique decomposition solution vectors. The size of T_a is $n^{O(\log r)}$, so our tensor decomposition algorithm will take $poly(n^{\log r})$ time to run. As long as r is a small constant (not scaling with n), this may still be computed in polynomial time. To empirically estimate each entry of the moment tensor, we count the number of times it occurs. This will give us a $1-\delta$ confidence interval that contains the correct moment value (w.p. $< \delta$, it deviates from the true value by $> \epsilon$):

$$p_{\ell_{i_1},\ldots,\ell_{i_d}} = \frac{|\{\langle \boldsymbol{s}, a, \boldsymbol{s'}\rangle \in \mathcal{T}(a) : s'_{\ell_{i_1}}, \ldots, s'_{\ell_{i_d}}, s_{\neg\ell_{i_1}}, \ldots, s_{\neg\ell_{i_d}}\}|}{|\{\langle \boldsymbol{s}, a, \boldsymbol{s'}\rangle \in \mathcal{T}(a) : s_{\neg\ell_{i_1}}, \ldots, s_{\neg\ell_{i_d}}\}|} \pm \Delta(\epsilon, \delta), \tag{2}$$

where $\Delta(\epsilon, \delta)$ is a small positive number depending on ϵ and δ. We will take the empirical count (mid-point of the interval) as our entry for the estimated tensor T_a (see appendix for a discussion). The caveat of this is that our tensor may have many missing entries. For the entry that corresponds to $\ell_{i_1}, \ldots, \ell_{i_d}$, we count the appearing frequency when all three literals are affected by this action. That is, we need to observe $s_{\neg\ell_{i_1}}, \ldots, s_{\neg\ell_{i_d}} = 1$, and $s_{\ell_{i_1}}, \ldots, s_{\ell_{i_d}} = 0$ in the previous state $\boldsymbol{s}$ changing into $s'_{\ell_{i_1}}, \ldots, s'_{\ell_{i_d}} = 1$, and $s'_{\neg\ell_{i_1}}, \ldots, s'_{\neg\ell_{i_d}} = 0$ in the next state $\boldsymbol{s'}$, due to the action a being taken. If such a pre-state never appears (or does not appear enough times), then we do not have a valid estimate, and this entry will be considered missing.

For example, consider a toy environment in which there are two Boolean fluents, top and left. Let's suppose there is an action updown that sets top to a uniform random value. The tensor has four 4×4 slices, but we will never encounter states with complementary fluents satisfied. These will be missing entries, which we'll mark as ?. (We do know they must be zero, but this helps illustrate the general problem.) The moments for the action updown are given in Fig. 1. Observe that when a literal ℓ is true in the pre-state, it does not matter whether ℓ is an effect of the action; as long as $\neg\ell$ is *not* an effect, ℓ will remain true. Thus, for any given state $\boldsymbol{s}$, if we consider the minor of the tensor T_a given by the indices of literals that are false in $\boldsymbol{s}$,[2] it is enough for this minor to be fully observed: the decomposition of the corresponding (minor of the) tensor identifies the distribution of post-states for the action a. Note that these minors of the tensor form blocks that are symmetrical w.r.t. the literals used as indices. For example, if the tensor is a matrix, then the effect set $\{\ell_i, \ell_j\}$

2. Generalizing from matrices, recall that a *minor* of a given tensor is another tensor obtained by omitting index values, thereby deleting some rows/columns/slices/etc.

is represented in the identical entries (ℓ_i, ℓ_j) and (ℓ_j, ℓ_i). Therefore, to obtain an accurate action model, it is sufficient to find a distribution of effects that is consistent with these minors of the tensor. Below, we will discuss how to decompose the tensor with many missing entries.

To guarantee safety, we add additional d-CNF preconditions that ensure that the agent only uses action a for these fully-observed minors: for each missing entry of the tensor T_a corresponding to the literals $\ell_{i_1}, \ldots, \ell_{i_d}$, we add a clause $\ell_{i_1} \vee \ldots \vee \ell_{i_d}$ to $pre(a)$. By De Morgan's law, this is equivalent to the condition that not all of $\ell_{i_1}, \ldots, \ell_{i_d}$ are 0, so for any state $\boldsymbol{s}$ satisfying the precondition, the minor for the literals that are false in $\boldsymbol{s}$ does not include this missing entry. Since entries are only considered missing when states for which all of $\ell_{i_1}, \ldots, \ell_{i_d}$ are 0 rarely occur in the training set, we will be able to guarantee that this additional precondition does not significantly reduce the completeness of the action model.

4. Effects Algorithm

Since there are missing entries in the data tensor T_a we construct for each action, we can not directly apply existing tensor decomposition algorithms to find a tensor decomposition to recover the effect vectors. Nevertheless, we saw that we only need to produce a distribution of effects that is consistent with the moments that we can estimate. Here, we will use this observation to develop an algorithm that first decomposes observed minors of the tensor, then combines these partial effect vectors to obtain a global effect vector. Our algorithm is Alg. 1, which works as follows:

We leverage the fact that these non-missing minors are symmetric to perform minor-wise decomposition, where the local solution of a minor is always consistent with a global solution v_i. Hence we can formulate constraints to make the these local pieces consistent with a solution v_i when they are pieced together. Because the full tensor is low rank, the minor must also be low rank. We can use existing methods to decompose them individually. In order to efficiently piece them together, we will iteratively pull out the top eigenvector from each block, and eliminate the vectors in other blocks that are inconsistent with it. Next round we will only pull out a top eigenvector that is consistent with the previous vectors. Combining them, we obtain a global effect vector.

4.1. Local decomposition algorithm

As long as we have uniqueness, we can use any suitable tensor decomposition method to obtain local 0-1 effect vectors from the minors. We will recall Jennrich's algorithm (Harshman, 1970; Leurgans et al., 1993) as the starting point for our combining method. We first reshape the higher degree tensor into degree-3 tensor. So, suppose we are given a tensor $T \in \mathbb{R}^{n^3}$, and it has decomposition $T = \sum_{i=1}^{r} \boldsymbol{a}_i^{\otimes 3}$ with orthogonal vectors $\boldsymbol{a}_i, \ldots, \boldsymbol{a}_r \in \mathbb{R}^n$. Then we can compute its this decomposition in the following steps:

First, pick a Gaussian random vector $\boldsymbol{g} \in \mathbb{R}^n$, and project T onto $\boldsymbol{g}$:

$$T_{\boldsymbol{g}} = \sum_{j=1}^{n} g_j T[j,:,:] = \sum_{j=1}^{n} \sum_{i=1}^{r} (g_j \cdot a_{i,j}) \cdot \boldsymbol{a}_i \otimes \boldsymbol{a}_i = \sum_{i=1}^{r} \langle \boldsymbol{g}, \boldsymbol{a}_i \rangle \cdot \boldsymbol{a}_i \otimes \boldsymbol{a}_i = \sum_{i=1}^{r} \langle \boldsymbol{g}, \boldsymbol{a}_i \rangle \cdot \boldsymbol{a}_i \boldsymbol{a}_i^{\top}.$$

Here $T[j,:,:]$ means the jth slice of tensor T. It can be viewed as a weighted sum of all the slices of matrices of the tensor T. This operation is called *contraction*. Then, we compute the singular value decomposition of $T_{\boldsymbol{g}}$. Since $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_r$ are orthogonal, they are a set of candidate eigenvectors of $T_{\boldsymbol{g}}$. Moreover, since $\boldsymbol{g}$ is Gaussian, the values $\langle \boldsymbol{g}, \boldsymbol{a}_i \rangle$ are distinct with probability 1, and so

input : $O(\log r)$-degree moment tensor where each entry is the empirical estimate of moments T_a, states $s \in \mathcal{T}(a)$.
output: global effect vectors $\{\boldsymbol{e}\}$ and their probabilities $\{p\}$
begin
  Reshape the moment tensor into degree-3 tensor.
  Draw a Gaussian vector $\boldsymbol{g}$ and contract the tensor blocks $B_{\boldsymbol{s}}$ for each $\boldsymbol{s} \in \mathcal{T}(a)$ to matrices using $\boldsymbol{g}$
  Compute the tensor decomposition for each block and obtain 0-1 local effect vectors $\hat{\boldsymbol{e}}_i$
  while *not all blocks have been reduced to zero* **do**
    while *not all blocks have been tightened and their constraints dropped* **do**
      Fix a new λ: geometric search for its upper bound, then binary search for tightness.
      Solve the SDP in Eq. 3,
      Find the tight blocks $B_{\boldsymbol{s}}$, and obtain their eigenvectors $\boldsymbol{x}_i$.
      Un-whiten each $\boldsymbol{x}_i$ to obtain $\boldsymbol{e}_i$, and eliminate all $\hat{\boldsymbol{e}}_{i'}$ that are inconsistent with $\boldsymbol{e}_i$ (as in Eq. 4.)
      Substitute the values for coordinates in $\boldsymbol{e}_i$ and drop the tight $B_{\boldsymbol{s}}$ $\boldsymbol{s'}$ constraints.
    end
    Combine un-whitened $\boldsymbol{e}_i$ vectors into a global effect vector $\boldsymbol{e}$.
    Collect $p = \lambda_m / \langle \boldsymbol{g}, \boldsymbol{e}_{i_m} \rangle$ as the probability for the current global effect, where $\lambda_m = \min_i \lambda_i$.
    Subtract $\lambda_m \boldsymbol{e}_i^{\otimes 2}$ from corresponding blocks.
  end
end

Algorithm 1: Stochastic Effect Learning

this singular value decomposition is unique. The computed eigenvectors must be $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_r$. In our case, $\boldsymbol{a}_i = \boldsymbol{e}_i$. (Note that Jennrich's algorithm requires the rank $r \leq n$, as we assume here.) However, this requires $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_r$ to be orthogonal. Following Sec. 2.3, the 0-1 effect vectors are linearly independent when the degree of the moments is high enough, so we can pre-process the tensor by whitening. That is, we will apply a whitening matrix W and compute:

$$T_{\boldsymbol{g}}^W = W T_{\boldsymbol{g}} W^\top = \sum_{i=1}^r \langle \boldsymbol{g}, \boldsymbol{e}_i \rangle \cdot W \boldsymbol{e}_i \boldsymbol{e}_i^\top W^\top = \sum_{i=1}^r \langle \boldsymbol{g}, \boldsymbol{e}_i \rangle \cdot (W \boldsymbol{e}_i)(W \boldsymbol{e}_i)^\top.$$

Here, abusing notation, we use $\boldsymbol{e}_i$ to also denote itself raised to tensor power $d = O(\log r)$, $flat(\boldsymbol{e}^{\otimes d})$, when the context is not confusing. Therefore, for the degree-2 moment matrix $M = \sum_{i=1}^r \boldsymbol{e}_i \boldsymbol{e}_i^\top$, we choose $W = M^{-1/2}$. One method of computing W is through PCA. So we will be computing vectors $\boldsymbol{a}_i = W \boldsymbol{e}_i$ by tensor decomposition. Then we can retrieve $\boldsymbol{e}_i$ by computing $\boldsymbol{e}_i = W^{-1} \boldsymbol{a}_i$. We will refer to this as un-whitening in the following discussion.

4.2. Composing the fragments

We will first sample a Gaussian vector $\boldsymbol{g}$ to contract all the blocks, and pre-compute the tensor decomposition for each block and obtain their unique local tensor decomposition vectors $\{\hat{\boldsymbol{e}}_i\}$. To obtain the global effect vector, we will extract one global eigenvector at a time, then subtract it from the blocks by a common weight, similar to eigendecomposition. We have a set of contracted blocks $\{B_{\boldsymbol{s}}\}_{\boldsymbol{s} \in \mathcal{T}(a)}$, where each $B_{\boldsymbol{s}} = (T|_{\ell:s_{\neg \ell}})^W_{\boldsymbol{g}|_{\ell:s_{\neg \ell}}}$ (with the abuse of notation $T = T|_{\ell:s_{\neg \ell}}$ referring to a local tensor block, and $\boldsymbol{g} = \boldsymbol{g}|_{\ell:s_{\neg \ell}}$ the corresponding projection of the global random Gaussian vector, when the context is clear). To obtain the global eigenvector, we extract one eigenvector from a block at a time, then eliminate inconsistent eigenvectors. To do that, we first vary the eigenvalue

bound λ and enforce the same lower bound λ for each block B. For a bound λ, for each coordinate i, we want $|(B\boldsymbol{x})_i| \geq \lambda |x_i|$, where $\boldsymbol{x}$'s are the variables of the program. When λ is tight for some block B, it must be the top eigenvalue for B and the projection of $\boldsymbol{x}$ on the coordinates in the block is an eigenvector of B. We use the Courant-Fischer formulation of the eigenvalue, written as a semidefinite program (SDP) as follows:

$$\max \lambda \; s.t. \;\text{ for all } \boldsymbol{s} \in \mathcal{T}(a) \qquad \langle U_{\boldsymbol{s}}, B_{\boldsymbol{s}} \rangle \geq \lambda \qquad U_{\boldsymbol{s}} \succeq 0 \qquad \text{and} \qquad \langle I, U_{\boldsymbol{s}} \rangle = 1 \tag{3}$$

We can factorize $U = V^\top DV$ for orthogonal matrices V where D is diagonal and $\langle U, U \rangle = 1$ enforces the diagonal to have norm 1. Then $\langle U, BU \rangle$ is maximized when V contains a top eigenvector v of B and D places all of the mass on the top eigenvector. We can also view this as having a common (big) matrix variable U with projections $P_{\boldsymbol{s}}$ selecting out each block $\boldsymbol{s}$ of U, where $P_{\boldsymbol{s}}$ selects the correct minor of U for a big block $\mathbf{B}$ that contains all the small blocks $B_{\boldsymbol{s}}$. We then have the constraints $\langle P_{\boldsymbol{s}} U P_{\boldsymbol{s}}^\top, \mathbf{B} P_{\boldsymbol{s}} U P_{\boldsymbol{s}}^\top \rangle \geq \lambda$.

To find the correct λ, we will keep increasing λ geometrically until the program is infeasible, i.e., λ is too large. Then we use binary search on this range to find λ that makes one block tight. After finding the tight block and its eigenvector $\boldsymbol{e}$, un-whiten it by applying W^{-1} and record this top eigenvector as a fragment of the top global eigenvector we want to retrieve. Then look through the pre-computed decomposed local vectors and find the vectors $\{\hat{\boldsymbol{e}}_{i'}\}$ that are inconsistent with the current tight vector. Then we subtract the weighted rank-1 matrix of each inconsistent $\hat{\boldsymbol{e}}_{i'}$ (contracted with the same Gaussian vector) from its corresponding block B':

$$B' - w_{i'} \langle \hat{\boldsymbol{e}}_{i'}, \boldsymbol{g} \rangle \hat{\boldsymbol{e}}_{i'}^{\otimes 2} = \sum_{\forall i \in [r], i \neq i'} w_i \langle \hat{\boldsymbol{e}}_i, \boldsymbol{g} \rangle \hat{\boldsymbol{e}}_i^{\otimes 2} \tag{4}$$

We then drop the constraints corresponding to the tight B, and we will continue the same procedure for the rest of the blocks. When all blocks have been tightened and dropped, we will collect all the un-whitened vector pieces together as one global vector $\boldsymbol{e}$, and subtract $\lambda_m \boldsymbol{e}_i^{\otimes 2}$ from each block, where $\lambda_m = \min_i \lambda_i$. We will collect $p = \lambda_m / \langle \boldsymbol{g}, \boldsymbol{e}_{i_m} \rangle$ as the probability for this global effect vector $\boldsymbol{e}$, where $i_m = \text{argmin}_i \lambda_i$. Then we will go back to the loop of tightening all the blocks again. We repeat this until all the blocks are reduced to zero. This completes Alg. 1. We illustrate the algorithm with an example execution in Appendix A.

To show the correctness of this algorithm, we need to show that the algorithm finds a consistent decomposition and terminates in polynomial time.

Lemma 2 *After each iteration of the inner loop, some $\hat{\boldsymbol{e}}_{i'}$ remains consistent with $\boldsymbol{e}_i$ in each block.*

Proof First, we note that if a post-whitening 0-1 effect vector $\boldsymbol{v}$ is not orthogonal to another post-whitening 0-1 effect vector $\boldsymbol{v'}$, then $\boldsymbol{v}$ and $\boldsymbol{v'}$ are consistent: Contrapositively, if $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ are distinct effect vectors in some block, we have chosen W so that $W\boldsymbol{e}_i$ and $W\boldsymbol{e}_j$ are orthogonal.

As in Jennrich's algorithm, the eigenvalues of the contracted blocks are distinct w.h.p. Recall that positive-semidefinite matrices are a conic combination of rank-1 matrices. Thus, for a fixed λ that is tight for a block, the solution to the SDP (3) is a rank-1 matrix on that block, which is the tensor square of some vector $\boldsymbol{v} = W\boldsymbol{e}_i$. For each other block, since the norm of the $U_{\boldsymbol{s}}$ matrix is 1, the $U_{\boldsymbol{s}}$ are convex combinations of tensor squares of vectors, where these $U_{\boldsymbol{s}}$ are consistent with $\boldsymbol{v}^{\otimes 2}$ on any common coordinates. In particular, as above, the whitening of the corresponding projection of $\boldsymbol{e}_j \neq \boldsymbol{e}_i$ onto the block gives a vector orthogonal to $W\boldsymbol{e}_i$. If the tensor $B_{\boldsymbol{s}}$ for

$\boldsymbol{s}$ only had support on components inconsistent with $U_{\boldsymbol{s}}$, we would have had $\langle U_{\boldsymbol{s}}, B\rangle = 0$, but $\langle U_{\boldsymbol{s}}, B\rangle \geq \lambda > 0$. Therefore, for each block $\boldsymbol{s}$, $U_{\boldsymbol{s}}$ must have positive weight on a rank-1 component that is the whitening of an effect vector consistent with $\boldsymbol{e}_i$. ■

Lemma 3 *The effect probabilities are consistent with each block for a state appearing in $\mathcal{T}(a)$.*

Proof Due to the uniqueness of the tensor decomposition, each weight w_i we compute is the same as its value in the latent tensor decomposition $\sum_{i=1}^{r} w_i \boldsymbol{e}_i^{\otimes d}$. Since the weight w_i is the probability, the local distribution is consistent. ■

Lemma 4 *The outer loop runs at most $r|\mathcal{T}(a)|$ iterations.*

Proof At the end of each outer iteration, we subtract $\lambda_m \boldsymbol{x}_{i_m}^{\otimes 2}$ from each block. We notice that λ_m is the eigenvalue and $\boldsymbol{x}_{i_m}$ is the corresponding eigenvector of A_{i_m}. Therefore, the eigenvector is eliminated from A_{i_m}'s spectral decomposition. At the end of each iteration of the outer loop, we eliminate at least one of the (at most) r eigenvectors of at least one of the (at most) $|\mathcal{T}(a)|$ blocks. Therefore, within $r|\mathcal{T}(a)|$ iterations, all blocks will be reduced to 0. ■

5. Safety and Completeness

We now establish the safety and completeness guarantees for Stochastic SAM. Due to space constraints, we analyze the quality of the tensor estimates in Appendix C.

Theorem 5 (Safety) *The distribution on trajectories in the true model M^* for any policy of length at most L is is ϵ-close to the Stochastic SAM model in total variation distance with probability $1-\delta$ given $m \geq \Omega(\frac{(Lr)^3|F|^{2\log r+1}|A|}{\epsilon^3} \log \frac{|F|^{\log r}|A|}{\delta})$ trajectories independently drawn from the planner on problems from $\mathcal{D}$. In particular, all actions that are applicable in a plan under the Stochastic SAM model are applicable to M^*.*

Proof We first argue that the transition distributions are close. By Lemma 10 (see Appendix C), we have that all of the non-missing entries of the empirical tensor are $(1 \pm \epsilon')$-close to the true moment tensor for M^*, so in particular, all of the fully-observed minors lie between the corresponding rescaled minors of M^*, which have a unique minimum-rank representation by Kruskal's Theorem and Lemma 1 (for degree $d = O(\log r)$), and indeed have the same $\leq r$ rank-1 components with the coefficients scaled by $(1+\epsilon')$ and $(1-\epsilon')$, respectively. The analysis of Section 4 shows that if the algorithm were run with these rescaled tensors for M^*, the algorithm would return M that recovers the rescaled transition probabilities exactly. The algorithm obtains the components of M by linear operations on the moment tensor. The algorithm therefore recovers a distribution on effect vectors from the empirical tensor such that each of the $\leq r$ effect vectors of each action a that has probability at least ϵ' occurs with probability that is $(1 \pm \epsilon')$-close to that of M^*.

Recall that the action preconditions for M only permits an action a to be taken in a state $\boldsymbol{s}$ if none of the tuples $\neg\ell_1, \ldots, \neg\ell_d$ true in $\boldsymbol{s}$ are marked as missing, so the corresponding minor of the tensor has no missing entries. Since the literal ℓ_i can only change value in $\boldsymbol{s'}$ if it is an effect and $s_{\neg\ell_i} = 1$, the distribution on $\boldsymbol{s'}$ given $\boldsymbol{s}$ and a is determined by the effect vectors in the tensor

decomposition of this minor (recalling the decomposition is unique), and in particular the next state $\boldsymbol{s}'$ is determined by the effect vector, which has probability given by the scaling of the corresponding component in the decomposition. Hence, either $\Pr_{M^*}[\boldsymbol{s}'|\boldsymbol{s}, a] < \epsilon'$ or

$$(1-\epsilon') \Pr_{M^*}[\boldsymbol{s}'|\boldsymbol{s}, a] \leq \Pr_{M}[\boldsymbol{s}'|\boldsymbol{s}, a] \leq (1+\epsilon') \Pr_{M^*}[\boldsymbol{s}'|\boldsymbol{s}, a].$$

Now, recall that the total variation distance between $\Pr_{\pi,M}$ and $\Pr_{\pi,M^*}$ may be expressed as $\max_{\text{sets of trajectories } S} |\Pr_{\pi,M}[S] - \Pr_{\pi,M^*}[S]|$. It therefore suffices to bound this difference for all events (sets of trajectories) S. By a union bound over the (at most) L steps and r possible effects, the total probability of any trajectory with a transition that has $\Pr_{M^*}[\boldsymbol{s}'|\boldsymbol{s}, a] < \epsilon'$ is at most $Lr\epsilon'$. For any given set of trajectories S, let S' be the subset that do not use any such low probability transitions. Then:

$$\Pr_{\pi,M}[S'] \leq \sum_{k=0}^{L} \sum_{\substack{\text{trajectories } T \in S' \text{of} \\ \pi \text{ running for } k \text{ steps}}} (1 + k\epsilon' + (k\epsilon')^2) \Pr_{\pi,M^*}[T] \leq (1+\epsilon) \Pr_{\pi,M^*}[S']$$

where we used $(1+\epsilon')^k \leq e^{\epsilon' k} \leq 1 + (\epsilon' k) + (\epsilon' k)^2$ for $\epsilon' k \leq 1$; note here that we will choose $\epsilon' \leq \frac{\epsilon}{4Lr}$ so that $k\epsilon' \leq \epsilon/4 < 1$ for all $k \leq L$, so we can invoke this inequality, and $(\epsilon/4)^2 < \epsilon/4$. By a similar calculation we also obtain that $\Pr_{\pi,M}[S'] \geq (1-\epsilon) \Pr_{\pi,M^*}[S']$; since the total probability of the trajectories with rare transitions is also at most $\epsilon/4$, $|\Pr_{\pi,M}[S] - \Pr_{\pi,M^*}[S]| \leq \epsilon$ as claimed.

Finally, since no transition that violates a precondition of any action a is ever observed, the preconditions of M include the literals of the precondition of M^*, and hence any action that is applicable in M is also applicable in M^*; so as long as we reach the same states in M^* as in a trajectory of M, the actions will remain applicable. ■

Theorem 6 (Approximate Completeness) *Suppose that for the distribution $\mathcal{D}$ over problems in a domain D and given planner, the problem is solved with probability p in L steps in expectation*[3] *over the draw from $\mathcal{D}$ and the planner itself. Given $m \geq \Omega(\frac{(Lr)^3|F|^{2\log r+1}|A|}{\epsilon^6} \log \frac{|F|^{\log r}|A|}{\delta})$ trajectories independently drawn from the planner on problems from $\mathcal{D}$, with probability $1-\delta$, the action model we learn satisfies the following: for a problem Π is sampled from $\mathcal{D}$, a policy that maximizes the probability of solving Π with length $L' = L/\epsilon$ solves Π with probability at least $p - \epsilon$ (over both the draw of Π and execution in M^*).*

Proof We first note that for $L' = L/\epsilon$, by Markov's inequality the probability that a policy that runs for L steps in expectation runs for more than L' steps with probability at most ϵ. For any policy π in the support of the planner on D (the distribution on policies used to sample the training trajectories), let π' be obtained from π by terminating if it runs for more than L' steps, π would take an action that does not respect the preconditions of M, or if, following an action, it observes a transition such that $\Pr_{M^*}[\boldsymbol{s}'|\boldsymbol{s}, a] < \epsilon' \leq \frac{\epsilon}{L'r}$. Recall that since π respected the preconditions of M^*, π only violates the preconditions of M if it would take an action in a state where there is a missing entry in the tensor, which only occurs for $\Pr_D[s_{\neg\ell_1}, \ldots, s_{\neg\ell_d}, a] < \mu_{min} = \frac{\epsilon'}{|A||F|^d}$. By a union bound over the tuples and actions, the total probability of π encountering such a missing entry in its trajectory is

3. If this is a worst-case bound instead, the dependence on ϵ may be reduced from ϵ^6 to ϵ^3.

at most ϵ'. Likewise, by a union bound over the length of the plan and number of effects in M^*, the probability of encountering such a rare transition is at most $L'r\epsilon' = \epsilon$. We note that π' is a legal policy for M, and moreover, in the distribution on training trajectories given by (π, D) on M^*, π only deviates from π' with total probability 3ϵ. Since the training trajectories sampled from (π, D) are assumed to succeed with probability p overall, the corresponding distribution(π', D) therefore also succeeds with probability at least $p - 3\epsilon$ in M^*. By Theorem 5 (using L' as the horizon bound), with probability $1 - \delta$ all policies running for L' steps are ϵ-close. Therefore, in particular, the corresponding π' also succeed at D with probability at least $p - 4\epsilon$ overall in M. Since the corresponding π' are solutions in M attaining this success probability overall and the planner is assumed to maximize the probability of success on D in M, it produces policies π'' with at least this probability of success in M on problems sampled from D. Because the distribution on trajectories in M and M^* is ϵ-close by Theorem 5 again, these policies actually attain success in M^* with probability at least $p - 5\epsilon$. By rescaling ϵ, we obtain the desired conclusion. ■

6. Conclusions and Future Work

We presented the first provably polynomial-time algorithm for learning stochastic action models that allows for the kind of correlations between effects that appear in most IPC probabilistic planning benchmarks, where the actions have a distribution on effects with small support (but possibly many effects). The main advantage over Juba and Stern (2022), which assumed that the effects on each fluent are mutually independent, lies in its ability to capture these highly correlated sets of effects. In the future, we intend to empirically evaluate the algorithm's performance relative to the existing methods that did not provide guarantees. (We include a prototype in the supplemental material.) The extension of our method to lifted representations (Juba et al., 2021) enables our method to be applied to the IPC PPDDL benchmarks, in spite of our use of semidefinite programming. The main barrier here is actually that stochastic planning remains very challenging. For example, while the Probabilistic Fast Downward planner (Steinmetz et al., 2016) is capable of solving the PPDDL benchmarks, it does not output a compact policy representation, which complicates the evaluation in a setting where the "ground truth" domain may not be identical to the PPDDL model.

We observe that our algorithm does not require observing complete states in the example trajectories, and in future work we will extend it to a partial information model. In contrast to algorithms for safe learning of deterministic environments, we do not handle conditional effects or numeric attributes (Mordoch et al., 2024, 2023), which are interesting directions. The main drawback of the algorithm is that solving large semidefinite programs, while polynomial time, is computationally expensive (cf. the current state of the art, Huang et al. (2022), is slightly worse than n^4 time). Thus, the main open question is whether we really need to solve a semidefinite program to compute the simultaneous top eigenvectors. In practice, the power method is usually a much faster method to compute top eigenvectors, and we suspect that along the lines of Anandkumar et al. (2014), such an algorithm should be much more efficient. The question is, how to handle the missing entries?

Acknowledgments

This work was supported by NSF awards IIS-1942336 and IIS-1939677. Part of this work was performed while BJ was visiting the Simons Institute for the Theory of Computing. We thank R. Stern for helpful conversations and our anonymous reviewers for their constructive suggestions.

References

Elizabeth S Allman and John A Rhodes. The identifiability of covarion models in phylogenetics. *IEEE/ACM transactions on computational biology and bioinformatics*, 6(1):76–88, 2008.

Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *Journal of machine learning research*, 15: 2773–2832, 2014.

Avrim L. Blum and John C. Langford. Probabilistic planning in the graphplan framework. In Susanne Biundo and Maria Fox, editors, *Recent Advances in AI Planning (ECP 1999)*, volume 1908 of *LNAI*, pages 319–332. Springer, Berlin, Heidelberg, 2000.

Emmanuel Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Communications of the ACM*, 55(6):111–119, jun 2012.

Sitan Chen and Ankur Moitra. Beyond the low-degree algorithm: mixtures of subcubes and their applications. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 869–880, 2019.

Yinyin Chen, Steven Culpepper, and Feng Liang. A sparse latent class model for cognitive diagnosis. *Psychometrika*, 85(1):121–153, 2020.

Steven Andrew Culpepper. An exploratory diagnostic model for ordinal responses with binary attributes: Identifiability and estimation. *Psychometrika*, 84(4):921–940, 2019.

Guanhua Fang, Jingchen Liu, and Zhiliang Ying. On the identifiability of diagnostic classification models. *Psychometrika*, 84:19–40, 2019.

Guanhua Fang, Jinxin Guo, Xin Xu, Zhiliang Ying, and Susu Zhang. Identifiability of bifactor models. *Statistica Sinica*, 31:2309–2330, 2021.

Dibya Ghosh, Chethan Anand Bhateja, and Sergey Levine. Reinforcement learning from passive data via latent intentions. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 11321–11339. PMLR, 2023.

Walton C Gibson. *The method of moments in electromagnetics*. CRC press, 2021.

Yuqi Gu. Blessing of dependence: Identifiability and geometry of discrete models with multiple binary latent variables. *Bernoulli*, 31(2):948–972, 2025.

Alastair R Hall. *Generalized method of moments*. Oxford University Press, 2004.

Richard A. Harshman. Foundations of the parafac procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.

Baihe Huang, Shunhua Jiang, Zhao Song, Runzhou Tao, and Ruizhe Zhang. Solving sdp faster: A robust ipm framework and efficient implementation. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 233–244. IEEE, 2022.

Nan Jiang and Tengyang Xie. Offline reinforcement learning in large state spaces: Algorithms and guarantees. *Statistical Science*, 40(4):570–596, 2025.

Brendan Juba and Roni Stern. Learning probably approximately complete and safe action models for stochastic worlds. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, pages 9795–9804, 2022.

Brendan Juba, Hai S Le, and Roni Stern. Safe learning of lifted action models. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning*, pages 379–389, 2021.

Kaiqiang Ke, Qian Lin, Zongkai Liu, Shenghong He, and Chao Yu. Conservative offline goal-conditioned implicit V-learning. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 29591–29607. PMLR, 2025.

Joseph B Kruskal. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear algebra and its applications*, 18(2):95–138, 1977.

Leonardo Lamanna and Luciano Serafini. Action model learning from noisy traces: a probabilistic approach. In *Proceedings of the 34th International Conference on Automated Planning and Scheduling*, pages 342–350, 2024.

Sue E. Leurgans, Robert T. Ross, and Rebecca B. Abel. A decomposition for three-way arrays. *SIAM Journal on Matrix Analysis and Applications*, 14(4):1064–1083, 1993.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Michael L. Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998.

Jiayuan Mao, Tomás Lozano-Pérez, Josh Tenenbaum, and Leslie Kaelbling. Pdsketch: Integrated domain programming, learning, and planning. *Advances in Neural Information Processing Systems*, 35:36972–36984, 2022.

David Martínez, Tony Ribeiro, Katsumi Inoue, Guillem Alenyà Ribas, and Carme Torras. Learning probabilistic action models from interpretation transitions. In *Proceedings of the 31st International Conference on Logic Programming (ICLP 2015)*, pages 1–14, 2015.

David Martínez, Guillem Alenya, Carme Torras, Tony Ribeiro, and Katsumi Inoue. Learning relational dynamics of stochastic domains for planning. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling*, pages 235–243, 2016.

László Mátyás et al. *Generalized method of moments estimation*, volume 5. Cambridge University Press, 1999.

Argaman Mordoch, Brendan Juba, and Roni Stern. Learning safe numeric action models. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, pages 12079–12086, 2023.

Argaman Mordoch, Enrico Scala, Roni Stern, and Brendan Juba. Safe learning of PDDL domains with conditional effects. In *Proceedings of the 34th International Conference on Automated Planning and Scheduling*, pages 387–395, 2024.

Kira Mourão, Luke Zettlemoyer, Ronald P. A. Petrick, and Mark Steedman. Learning STRIPS operators from noisy and incomplete observations. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence*, pages 614–623, 2012.

Vivek Myers, Bill Zheng, Benjamin Eysenbach, and Sergey Levine. Offline goal-conditioned reinforcement learning with quasimetric representations. In *Advances in Neural Information Processing Systems 38*, pages 19654–19679, 2026.

Whitney K Newey and Kenneth D West. Hypothesis testing with efficient method of moments estimation. *International Economic Review*, pages 777–787, 1987.

Michel M Ney. Method of moments as applied to electromagnetic problems. *IEEE transactions on microwave theory and techniques*, 33(10):972–980, 1985.

Masao Ogaki. Generalized method of moments: Econometric applications. In *Handbook of Statistics*, pages 455–487. Elsevier, 1993.

Seohong Park, Dibya Ghosh, Benjamin Eysenbach, and Sergey Levine. Hiql: Offline goal-conditioned RL with latent states as actions. In *Advances in Neural Information Processing Systems 36*, pages 34866–34891, 2023.

Hanna M. Pasula, Luke S. Zettlemoyer, and Leslie Pack Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29:309–352, 2007.

Karl Pearson. Method of moments and method of maximum likelihood. *Biometrika*, 28(1/2):34–59, 1936.

Scott Sanner. Relational dynamic influence diagram language (rddl): Language description. Unpublished manuscript, 2010.

Tselil Schramm and David Steurer. Fast and robust tensor decomposition with applications to dictionary learning. In Satyen Kale and Ohad Shamir, editors, *Proceedings of the 2017 Conference on Learning Theory*, volume 65 of *Proceedings of Machine Learning Research*, pages 1760–1793. PMLR, 07–10 Jul 2017.

Marcel Steinmetz, Jörg Hoffmann, and Olivier Buffet. Goal probability analysis in probabilistic planning: Exploring and enhancing the state of the art. *Journal of Artificial Intelligence Research*, 57:229–271, 2016.

Mianchu Wang, Rui Yang, Xi Chen, Hao Sun, Meng Fang, and Giovanni Montana. Goplan: Goal-conditioned offline reinforcement learning by planning with learned models. *Transactions on Machine Learning Research*, 2024.

	top	¬top	left	¬left
top	0	?	0	0
¬top	?	?	?	?
left	0	?	0	?
¬left	0	?	?	0

(*a*) top slice, iter. 2

	top	¬top	left	¬left
top	?	?	?	?
¬top	?	$1/2$	0	0
left	?	0	0	?
¬left	?	0	?	0

(*b*) ¬top slice, iter. 2

	top	¬top	left	¬left
top	0	?	0	?
¬top	?	0	0	?
left	0	0	0	?
¬left	?	?	?	?

(*c*) left slice, iter. 2

	top	¬top	left	¬left
top	0	?	?	0
¬top	?	0	?	0
left	?	?	?	?
¬left	0	0	?	0

(*d*) ¬left slice, iter. 2

Figure 2: Residual third moment tensor on algorithm's second iteration, with one outcome removed

Jeffrey M Wooldridge. Applications of generalized method of moments estimation. *Journal of Economic perspectives*, 15(4):87–100, 2001.

Gongjun Xu. Identifiability of restricted latent class models with binary responses. *The Annals of Statistics*, 45(2):675–707, 2017.

Chengrun Yang, Lijun Ding, Ziyang Wu, and Madeleine Udell. Tenips: Inverse propensity sampling for tensor completion. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3160–3168. PMLR, 2021.

Håkan L. S. Younes and Michael L. Littman. PPDDL1.0: The language for the probabilistic part of IPC-4. In *Proceedings of the International Planning Competition*, 2004.

Yaocheng Zhang, Yuanheng Zhu, Yuqian Fu, Songjun Tu, and Dongbin Zhao. Offline goal-conditioned reinforcement learning with elastic-subgoal diffused policy learning. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, pages 2336–2344, 2025.

Hankz Hankui Zhuo and Subbarao Kambhampati. Action-model acquisition from noisy plan traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 2444–2450, 2013.

Appendix A. An Example of the Effects Learning Algorithm

Recall our toy environment from Fig. 1. We walk through our algorithm run on this partial tensor.

The algorithm uses a random contraction, which we will suppose is a sum over distinct slices; note that the observed blocks only contain one literal per each fluent, and so the contractions we consider only add a combination of one of the first two slices to one of the second two slices, both of which are all 0 in the observed entries. So, the contracted observed blocks are rescaled minors of the first two slices.

Suppose that the algorithm first encounters the partial effect vector $[1\ ?\ 0\ ?]$, which has eigenvalue $1/2$. Since the effect of top rules out the effect ¬top, we also fix the second component to 0. This eliminates the ¬top slice. Now, with the observed block given by top and ¬left, we find that the ¬left effect must have value 0 – actually, this block must also be tight with eigenvalue $1/2$ – so we obtain a partial effect vector $[1\ 0\ 0\ 0]$ with eigenvalue $1/2$, corresponding to the effect that sets

top true with probability $1/2$. We subtract the tensor cube $[1\ 0\ 0\ 0]^{\otimes 3}$ weighted by $1/2$ from the tensor to obtain the tensor in Figure 2.

Now the nontrivial constraints correspond to vectors in the ¬top slice, e.g., the minor of that slice with ¬top and left, which gives the vector $[?\ 1\ 0\ ?]$ with eigenvalue $1/2$. Again, the effect not top rules out the effect top, so this must be extended to $[0\ 1\ 0\ ?]$, and we eliminate the top slice, and the ¬top and ¬left minor, where the eigenvector assigns ¬left 0, is also tight with eigenvalue $1/2$. Thus, we obtain the vector $[0\ 1\ 0\ 0]$ with eigenvalue $1/2$. Subtracting this off finally gives an all-0 tensor. We thus obtain that the effects are top with probability $1/2$, and ¬top with probability $1/2$.

Appendix B. Linear Independence Among Tensor Powers of Boolean Vectors

We begin by identifying the smallest linear dependences among each tensor power of Boolean vectors.

Theorem 7 *For $k \in \mathbb{N}$, let $S \subseteq \{0,1\}^{k+1}$ of size $|S| < 2^{k+1} - 1$ be given. Then the set $S^{\otimes k} = \{\boldsymbol{x}^{\otimes k} : \boldsymbol{x} \in S\}$ is linearly independent; moreover, the only linear dependences for sets of size $2^{k+1} - 1$ are of the form $\sum_{\boldsymbol{x} \neq \mathbf{0}} \alpha(-1)^{w(\boldsymbol{x})} \boldsymbol{x}^{\otimes k}$ for $\alpha \neq 0$ where $w(\boldsymbol{x})$ denotes the Hamming weight of $\boldsymbol{x}$, $\sum_i x_i$.*

Proof We show this by induction on k. For $k = 1$, $|S| \leq 2$, and indeed for any two distinct $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^2$, without loss of generality we have in some coordinate i $x_i = 1$ and $y_i = 0$. Then in any linear combination in which $\boldsymbol{x}$ has nonzero weight α, the ith coordinate is $\alpha \cdot 1 \neq 0$, so $S^{\otimes 1} = S$ is indeed linearly independent. For $|S| = 3$, the vector $(0,0)$ would yield a linear dependence among two vectors which we see is impossible. Thus the set must be $\{(1,1),(1,0),(0,1)\}$. Suppose that $(1,1)$ has weight $\alpha \neq 0$. Then $(1,0)$ and $(0,1)$ must have weight $-\alpha$ to yield a linear dependence.

Given the claim holds for $k-1$, we show it for k. Consider any set S for which for some coordinate i^*, more than $2^k - 1$ of the members $\boldsymbol{x} \in S$ have $x_{i^*} = 0$. Consider any linear combination of $S^{\otimes k}$, $\sum_{\boldsymbol{x} \in S} \alpha_x \boldsymbol{x}^{\otimes k}$. Then in the slice for i^* of $S^{\otimes k}$, we would have the linear combination $\sum_{\boldsymbol{x} \in S : x_{i^*} = 1} \alpha_x \boldsymbol{x}_{-i^*}^{\otimes k-1}$; since $|\{\boldsymbol{x} \in S : x_{i^*} = 1\}| \leq 2^k - 2$ by hypothesis, the induction hypothesis yields that these are linearly independent and the linear combination must be nonzero.

Now suppose some coordinate i^* has $2^k - 1$ 1s. Supposing that each of the slices have a linear dependence, we see that the coefficient of $\boldsymbol{x}_{-i^*}^{\otimes k-1}$ in the linear dependence must be $\alpha(-1)^{w(\boldsymbol{x}_{-i^*})}$ by induction hypothesis. But then, we see that in the $(i^*)^k$ entry of the tensor, we must obtain $\sum_{w=1}^{k} \alpha(-1)^w \binom{k}{w} = \alpha((1-1)^k - 1) \neq 0$. Thus, any linear dependence must be for a set of vectors that has 2^k 1s in each coordinate.

But now we observe that we cannot have sufficiently high weight in all coordinates to obtain a linear dependence: indeed, to have at least 2^k 1s in all $k+1$ coordinates, we would have total weight at least $(k+1)2^k$. But, in dimension $k+1$, observe that the greatest weight of $2^{k+1} - 2$ distinct vectors is obtained by omitting the vector $\mathbf{0}$ and some weight-1 vector, which has total weight at most $\sum_{w=1}^{k+1} w \cdot \binom{k+1}{w} - 1 = (k+1)2^k - 1$ (indeed, recall, $\sum_{w=1}^{k+1} x^w \binom{k+1}{w} = (1+x)^{k+1} - 1$, and taking derivatives, $\sum_{w=1}^{k+1} w x^{w-1} \binom{k+1}{w} = (k+1)(1+x)^k$). Thus there is no linear dependence among $2^{k+1} - 2$ vectors.

For $2^{k+1} - 1$ vectors, we again must use all vectors but $\mathbf{0}$. We note that in each coordinate i, we must thus use the standard basis vector $\boldsymbol{e}^{(i)}$, for which $\boldsymbol{e}_{-i}^{(i)} = \mathbf{0} \in \{0,1\}^k$. Thus, we see that the slices for each ith coordinate have exactly $2^k - 1$ nonzero vectors participating, and hence by

induction hypothesis have weights $\alpha(-1)^{w(\boldsymbol{x}_{-i})-1}$. To obtain a linear dependence, now, we see that $\boldsymbol{e}^{(i)}$ must have weight $-\alpha = \alpha(-1)^{w(\boldsymbol{e}^{(i)})}$ so that $\alpha + \sum_{w=1}^{k} \alpha(-1)^{w+1}\binom{k}{w} = -\alpha - \alpha((1-1)^k - 1) = 0$, as claimed. ■

Now we observe that for Boolean tensors, increasing the dimension cannot yield a smaller linearly dependent set; this yields our final claim in this section, Lemma 1.

Lemma 8 (1) *For all $k, n \in \mathbb{N}$ and $S \subseteq \{0,1\}^n$, if $|S| \leq 2^{k+1} - 2$, then $S^{\otimes k}$ is linearly independent.*

Proof We proceed by induction on n. For $n < k+1$, we observe that we can embed S into $\{0,1\}^{k+1}$ by introducing 0 coordinates to each $\boldsymbol{x}$ to obtain a set of vectors S', where the tensors $\boldsymbol{x}^{\otimes k}$ for $\boldsymbol{x} \in S$ are obtained on the minors of $\boldsymbol{x'}^{\otimes k}$ for $\boldsymbol{x'} \in S'$. Then for all $n \leq k+1$, the claim follows from Theorem 7.

Supposing now that we have established the claim for $n - 1 \geq k + 1$, we proceed to show it for n as follows: Suppose we have a linear dependence in S of size at most $2^{k+1} - 2$, and consider each of the coordinate-wise projections $S_{-i} = \{\boldsymbol{x}_{-i} : \boldsymbol{x} \in S\}$. Since S_{-i} has dimension $n-1$, we know by our induction hypothesis that S_{-i} is linearly independent. Therefore, in our linear dependence, we must have that for each $\boldsymbol{x'} \in \{0,1\}^{n-1}$ in the image of the projection, there must be more than one member of S such that the total weight in the linear combination sums to 0. But, since S is a set of Boolean vectors, there are exactly two vectors in $\{0,1\}^n$ that map to each $\boldsymbol{x'} \in \{0,1\}^{n-1}$; we thus find that if one is present in S, the other must be as well so that they may cancel. But now, we see that indeed, for any $\boldsymbol{x} \in S$, all of the vectors $\boldsymbol{y}$ of Hamming distance 1 from $\boldsymbol{x}$ are also in S. Indeed, since all of $\{0,1\}^n$ can be reached by a series of Hamming distance 1 neighbors from any member, S must contain all of $\{0,1\}^n$. Then $|S| = 2^n \geq 2^{k+2} > 2^{k+1} - 2$, a contradiction. ■

Appendix C. Convergence of Empirical Estimates

In this section we establish Lemma 10: with probability $1 - \delta/(2|F|^d|A|)$, the learned effect probability $\hat{p}$ is close to the true p^* in the sense that $(1-\epsilon)\hat{p} \leq p^* \leq (1+\epsilon)\hat{p}$. This guarantees the effect probabilities we learn are close to the true effect probabilities. We use $\mu_{min} = \frac{\epsilon'}{|A||F|^d}$ (with ϵ' fixed in the proof of Theorem 5) as our minimum probability of observing a tuple, below which we will (aim to) mark it as missing in the tensor. We first do a calculation with Hoeffding's bound to estimate the number of trajectories we will need to sample to guarantee that we have the desired number of observations of the given tuple:

Lemma 9 *A Binomial(m, p) random variable such that $p \geq \mu_{min}$ takes value at least q with probability $1 - \delta$ for $m \geq \frac{3}{2}\mu_{min} s + 3\ln(1/\delta)$*

Proof In pq trials, we have at least q successes in expectation; we will calculate the number of additional trials a (so: $m = pq + a$) needed to guarantee at least q with probability $1 - \delta$. Plugging in Hoeffding's inequality, we find that a satisfying

$$\exp\left(-2(pq+a)\left(\frac{a}{2(pq+a)}\right)^2\right) = \exp\left(\frac{-a^2}{2(pq+a)}\right) \leq \delta$$

suffices. As $a > 0$ increases, the LHS decreases, so we merely need to find the smallest a that suffices. We note that equality holds when

$$\ln \frac{1}{\delta} = \frac{a^2}{2(pq+a)}$$
$$0 = a^2 - 2a \ln \frac{1}{\delta} - 2pq \ln \frac{1}{\delta}$$

which has the solutions

$$a = \frac{2\ln\frac{1}{\delta} \pm \sqrt{4\ln^2\frac{1}{\delta} + 8pq\ln\frac{1}{\delta}}}{2} = \ln\frac{1}{\delta} \pm \sqrt{\ln^2\frac{1}{\delta} + 2pq\ln\frac{1}{\delta}}.$$

To simplify the expression for a, we recall that $\sqrt{x^2+y^2} \leq x+y$ and $\sqrt{xy} \leq (x+y)/2$ for non-negative x and y, and obtain:

$$\begin{aligned} a &= \ln\frac{1}{\delta} + \sqrt{\ln^2\frac{1}{\delta} + 2pq\ln\frac{1}{\delta}} \\ &\leq \ln\frac{1}{\delta} + \ln\frac{1}{\delta} + \sqrt{pq\ln\frac{1}{\delta}} \\ &\leq \ln\frac{1}{\delta} + \ln\frac{1}{\delta} + \frac{1}{2}pq + \ln\frac{1}{\delta} \end{aligned}$$

Therefore, $\frac{3}{2}\mu_{min}q + 3\ln(1/\delta)$ trials suffice. ■

In the following, we will use the notation $\#_a$ to denote the number of $(\boldsymbol{s}, a, \boldsymbol{s'})$ triplets in the training set (with the indicated action a) for which the specified condition holds. Let $x = \Pr[s'_{\ell_1}, \ldots, s'_{\ell_d} | s_{\neg\ell_1}, \ldots, s_{\neg\ell_d}, a]$ be the true moment. Let $\hat{x} = \frac{\#_a(s'_{\ell_1},\ldots,s'_{\ell_d},s_{\neg\ell_1},\ldots,s_{\neg\ell_d})}{\#_a(s_{\neg\ell_1},\ldots,s_{\neg\ell_d})}$ be the empirical estimate of x. Now, using the above lemma, we guarantee $\hat{x}$ is close to x:

Lemma 10 *Given a sample of* $3\left(\frac{1}{\epsilon'^3}|F|^d|A|+1\right)\ln(2|F|^d|A|/\delta)$ *trajectories, with probability* $1-\delta$*, for all actions* $a \in A$ *and tuples* $\ell_1, \ldots, \ell_d$ *such that* $\Pr_{\pi,D}[\exists \boldsymbol{s}, a : s_{\neg\ell_1}, \ldots, s_{\neg\ell_d}, a] \geq \mu_{min}$*, and* $\Pr[s'_{\ell_1}, \ldots, s'_{\ell_d} | s_{\neg\ell_1}, \ldots, s_{\neg\ell_d}, a] \geq \epsilon'$*,*

$$\left|\Pr[s'_{\ell_1}, \ldots, s'_{\ell_d} | s_{\neg\ell_1}, \ldots, s_{\neg\ell_d}, a] - \hat{x}\right| \leq \epsilon' \Pr[s'_{\ell_1}, \ldots, s'_{\ell_d} | s_{\neg\ell_1}, \ldots, s_{\neg\ell_d}, a]$$

Proof Note that in a sample of trajectories, for any given tuple $\#_a(s'_{\ell_1}, \ldots, s'_{\ell_d}, s_{\neg\ell_1}, \ldots, s_{\neg\ell_d})$ may be written as a sum of indicator random variables, where in particular these counts change by at most 1 on each transition, and the number of observations $\#_a(s_{\neg\ell_1}, \ldots, s_{\neg\ell_d})$ reaching a given value is a valid stopping time (i.e., depends only on observations up to that point in the trajectory). We may therefore apply the Azuma-Hoeffding inequality to the empirical fraction $\hat{x}$, and obtain $|x - \hat{x}| \leq \sqrt{\frac{\ln(2|F|^d|A|/\delta)}{2\#_a(s_{\neg\ell_1},\ldots,s_{\neg\ell_d})}}$ with probability $1 - \frac{\delta}{2|F|^d|A|}$. By a union bound over all actions and tuples, now, the above bound holds simultaneously with probability $1 - \delta/2$. For those for which $x = \Pr[\ell_1, \ldots, \ell_d | \neg\ell_1, \ldots, \neg\ell_d, a] \geq \mu_{min} = \frac{\epsilon'}{|A||F|^d}$, moreover, we now have

$$\sqrt{\frac{\ln(2|F|^d|A|/\delta)}{2\#_a(s_{\neg\ell_1}, \ldots, s_{\neg\ell_d})}} \cdot \frac{1}{x} \leq \sqrt{\frac{\ln(2|F|^d|A|/\delta)}{2\#_a(s_{\neg\ell_1}, \ldots, s_{\neg\ell_d})}} \cdot \frac{1}{\epsilon'}$$

so the desired bound holds as long as for all tuples with $\Pr_D[\exists \boldsymbol{s}, a : s_{\neg \ell_1}, \ldots, s_{\neg \ell_d}, a] \geq \mu_{min}$ we have

$$\#_a(s_{\neg \ell_1}, \ldots, s_{\neg \ell_d}) \geq 2\frac{1}{\epsilon'^2} \ln((2|F|^d|A|/\delta)).$$

It now follows by our earlier calculation that if

$$m \geq \frac{3}{2}\frac{1}{\mu_{min}} \cdot 2\frac{1}{\epsilon'^2} \ln(2|F|^d|A|/\delta) + 3\ln(2|F|^d|A|/\delta) = 3\left(\frac{1}{\epsilon'^3}|F|^d|A| + 1\right) \ln(2|F|^d|A|/\delta)$$

we obtain a sufficient number of observations of $a \in A$ and $\boldsymbol{s}$ with $s_{\ell_1}, \ldots, s_{\ell_d} = 1$ with probability $1 - \frac{\delta}{2|F|^d|A|}$. By another union bound, this gives a sufficient number of observations of all of these with probability $1 - \delta/2$. ■