Stochastic Safe Action Model Learning

Zihao Deng
Washington University in St. Louis
St. Louis, MO 63130
`zihao.deng@wustl.edu`

Brendan Juba
Washington University in St. Louis
St. Louis, MO 63130
`bjuba@wustl.edu`

Abstract

Hand-crafting models of interactive domains is challenging, especially when the dynamics of the domain are stochastic. Therefore, it's useful to be able 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 state attributes (a.k.a. fluents). This class captures the benchmark domains used in stochastic planning, in contrast to the prior work that assumed independence of the effects on individual fluents. Our algorithm has polynomial time and sample complexity when the size of the support is bounded by a constant. Importantly, our learning 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 guaranteed to be 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.

1 Introduction

In classical (high-level) task planning problems, a domain model describes the the interaction between an environment and the planning agent. A 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. Such formal languages include STRIPS [Fikes and Nilsson, 1971] and PPDDL[Aeronautiques et al., 1998], 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. To overcome the modeling problem, many approaches have been proposed to automatically learn the domain model [Yang et al., 2007, Cresswell and Gregory, 2011, Zhuo and Kambhampati, 2013, Stern and Juba, 2017, Aineto et al., 2019, Juba et al., 2021, Juba and Stern, 2022].

This problem is even more difficult when the domain dynamics is stochastic [Juba and Stern, 2022]. Indeed, in a deterministic environment, we can learn that some state attributes are not part of the effect as long as they are not in the post transition state, and that they are part of an effect as long as it appears in the post state but not the previous state. However, when the effects are stochastic, it's possible that an effect does not appear in the post state and yet has significant probability of appearing. Consequently, we will need many observations; 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 domain models where effects simply set some fluents to take specific values, we may not observe whether or not an effect occurred if the fluent already had that value prior to taking the action. Therefore, in any given transition, we generally cannot know for certain which fluents would have been set by the random effects of the action, even when the values of the fluents are fully observed.

Our goal is to safely learn an 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. We also seek a guarantee that the model is sufficiently accurate to capture and avoid potential danger during planning. This is similar to offline reinforcement learning [Levine et al., 2020], except that we wish to learn a reusable model that allows solving many goals, and we learn high-level task planning representations (with concise PPDDL representations). To ensure accuracy, 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 possible transitions in the environment. 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 the assumption that the effects of actions on each fluent are independent random variables. In this work, we relax that assumption and provide the algorithm for a general stochastic environment.

We note that the prior work by Juba and Stern [2022] gave an efficient and safe method for learning the preconditions of the actions that carries over to this more general problem. So, we focus on the problem of learning the distribution of the random effects for each action a. Specifically we will use method of moments originally 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 may be efficiently realized using algorithms for tensor decomposition to fit a wide class of probabilistic models, given a certain "identifiability" assumption: that the moments uniquely determine the parameters. Typically, one assumes either linear independence of the vectors of outcomes in the support, or that the parameters are "generic" numbers (in the sense of algebraic geometry) to guarantee identifiability; unfortunately, numbers with finite representations are *not* "generic." Worse, in our setting, the full effects distribution actually may not be identifiable.

In this work, we overcome these challenges as follows: first, we observe that for the kinds of simple discrete effects used in classical planning models (e.g., as appearing 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 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. Third, we give a polynomial-time algorithm to construct a set of stochastic effects that is consistent with the observed marginals. Finally, we show that we can add some 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 argue that this does not significantly reduce the completeness of the model, relative to the distribution on example trajectories.

1.1 Related work

A few approaches to learning planning models for stochastic domains have been previously proposed. Pasula et al. [2007] formulated their learning problem as optimizing an objective that could not be tractably optimized, and proposed a greedy heuristic solution. Unfortunately, this heuristic cannot provide the soundness or approximate completeness guarantees that we seek. On the other hand, Mourão et al. [2012] assumes 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.) More recently, Mao et al. [2022] proposed to use a neural network model to predict missing parts of an action model. While presumably quite powerful, this again comes at the price of any expectation of soundness and prevents the representation from being used in standard planners.

Our algorithm for constructing consistent set of effects solves a problem resembling the analogue of low-rank matrix completion (e.g., Candès and Recht [2012]) for tensors with a nonuniform observation model. Prima facie, this is not possible since the solution may not be identifiable. As we discuss above, it is crucial for our problem that it is sufficient to only match the distributions on marginals for which we have observations. The "completed" entries may, in general, be wildly off, and so the additional preconditions are necessary to 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 ensures that at least some of our instances would not satisfy the assumptions required for those algorithms.

2 Preliminaries

We now recall our problem domain and the Stochastic Safe Action Model Learning (SAM) problem. Subsequently, we will introduce some mathematical tools for the method of moments.

2.1 Stochastic Safe Action Model learning Problem

We formulate our problem in terms of grounded PPDDL representations for simplicity. 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. 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 s is a vector of assignments of Boolean values $s(f)$ to each respective fluent f in F. We abuse notation by denoting the indicators $1[s(f) = 1]$ as f and $1[s(f) = 0]$ as $\neg f$, which we refer to as *literals*.

An action is an operation that can be taken by the planning agent to change the values in s, hence transitioning to another state s'. 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 s to allow the action a to be taken. 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.

We consider a fragment of PPDDL in which the effects have a single "`probabilistic`" block for each action: this means that for each action, there is a sequence of r partial assignments $e_1, e_2, \ldots, e_r$ where each e_i is associated with a respective probability p_i. That is, when action a is taken, with probability p_i, the partial assignment e_i will occur in the next state s', and any fluents not set in e_i remain the same value as in the previous state s. We assume that the number of distinct outcomes $r = r(a)$ for each a is at most a small constant, which is consistent with the benchmarks used in the IPC probabilistic tracks, where the number of sets of outcomes is generally below five. We stress that each outcome may set any number of fluents. Note that the action model thus specifies the probability of transitioning from a state s to another state s' by applying a, denoted by $Pr_M[s'|a, s]$.

A PPDDL planning problem $\Pi = \langle D, s_I, s_G \rangle$ consists of a domain D, a starting state s_I, and a goal state s_G. A solution to a PPDDL planning problem is a policy π, which is a mapping from the states to the actions $\pi : 2^F \to A$. 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 $\pi(s_I)$ at the starting state s_I. The execution ends when the goal state 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 $\langle s_0, a_0, s_1, a_1, \ldots, a_{|\mathcal{T}|}, s_{|\mathcal{T}|} \rangle$. 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 $s_0 = s_I$. The length $|\mathcal{T}|$ of trajectory $\mathcal{T}$ is the number of actions taken.

In the Stochastic Safe Action Model (SAM) Learning problem, we suppose that 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 pair of problem $\langle D, s_I^{(i)}, s_G^{(i)} \rangle$ and policy $\pi^{(i)}$ from the distribution, and then executing $\pi^{(i)}$ in D from $s_I^{(i)}$ to produce $\mathcal{T}_i$. We are 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 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 $\hat{D}$ in expectation, 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 problems $\langle D, s'_I, s'_G \rangle$ and policies π' sampled independently from the training distribution, with probability $1 - \epsilon$ there is a policy $\hat{\pi}$ that takes

legal actions in $\hat{D}$ and such that the probability that $\hat{\pi}$ reaches s'_G from s'_I in $\hat{D}$ is less than the probability that π' reaches s'_G from s'_I in D by at most ϵ.

In our guarantees, in addition to the running time of an algorithm for this problem, we must 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.

2.2 Tensor decomposition

A degree-d, dimension-n tensor T is an array of numbers $T[i_1, \dots, i_d]$ 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:

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

for vectors $v_k^{(j)}$ of dimension n. Equivalently, we can write it as: $T = \sum_{k=1}^{r} w_k v_k^{(1)} \otimes \cdots \otimes v_k^{(d)}$, where $\otimes$ is the outer product of two vectors. The minimum number of terms r in such possible decompositions is called the tensor rank of T. We refer to the set of vectors $V^{(j)} = [v_1^{(j)}, \dots, v_r^{(j)}]$ as the jth mode of this tensor decomposition, where $j \in [d]$. If all modes $V^{(j)}$ are the same $V = [v_1, \dots, v_r]$, we can write the decomposition as

$$T = \sum_{k=1}^{r} w_k v_k^{\otimes d}, \tag{1}$$

where $v_k^{\otimes d}$ is the outer product of a vector v_k with itself d times. Note that for positive w_k (or odd d), by rescaling each vector v_k by $w_k^{1/d}$, we can obtain the same tensor while dropping w_k.

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, 2022, 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 $\sum_{k=1}^{r} a_k \otimes b_k \otimes c_k$. Let $A = [a_1, \dots, a_r]$, $B = [b_1, \dots, b_r]$, and $C = [c_1, \dots, c_r]$ 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 Tensor Decomposition for Boolean Components

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]. Suppose that we are given a degree-$2k+1$ tensor $\sum_{i=1}^{r} w_i v_i^{\otimes(2k+1)}$, where $v_i \in \{0,1\}^d$. The goal is to obtain v_i. We will re-shape it and obtain a degree-3 tensor $\sum_i^r w_i v_i \otimes flat(v_i^{\otimes k}) \otimes flat(v_i^{\otimes k})$. Here $flat(M)$ means that we rearrange the tensor into a vector.

By Kruskal's Theorem, we can argue that the tensor decomposition of $\sum_i^r v_i \otimes flat(v_i^{\otimes k}) \otimes flat(v_i^{\otimes k})$ is unique up to permutation. Indeed, suppose $V = \{v_1, \dots, v_r\}$, and $V^{\otimes k} = \{flat(v_1^{\otimes k}), \dots, flat(v_r^{\otimes k})\}$. According to Lemma 1, $V^{\otimes k}$ must be full rank when $k = O(\log(r))$:

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.*

We omit the proof due to space constraints. 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 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 Approach to Stochastic SAM Learning

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 $(s, a, s') \in T$, where $T \in \mathcal{T}$, we have that if a literal is not in the previous state then it cannot be a precondition. Precisely: $\forall \ell : s(\neg \ell) \Rightarrow \ell \notin pre(a)$ 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 $\langle s, a, s' \rangle$ triplets for action a. States s is the pre-state and s' is the post-state of action a. We (initially) set the preconditions of a to be $pre(a) = \{\ell : \forall \langle s, a, s' \rangle \in \mathcal{T}(a)\ s(\ell)\}$.

3.2 Learning effects

We now need to recover the set of effects $\{e_i\}$ for each action a, where each effect 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 will arise is that we may not have such states where a is taken and ℓ is false; we will return to this issue later.

For the purpose of 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: $e_1 = \{\ell_1, \ell_3\}$, $e_2 = \{\ell_1, \ell_2\}$, and $e_3 = \{\ell_2\}$. the probability values we observed and approximated are

$$p_{\ell_i,\ell_j,\ell_k} := Pr\left[s'(\ell_i), s'(\ell_j), s'(\ell_k) | s(\neg \ell_i), s(\neg \ell_j), s(\neg \ell_k), a\right]. \tag{2}$$

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} as in Equation 2 and indexed by three literals ℓ_i, ℓ_j, ℓ_k. Its value will be the *sum of all p_i for all effects 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 effects we want to recover can be formulated as 0-1 vectors:

$$e_1 = (1, 0, 1, 0),\ \ e_2 = (1, 1, 0, 0),\ \ e_3 = (0, 1, 0, 0)$$

The degree-3 moments can arranged into a degree-3 tensor as follows:

$$T_a = \sum_{i=1}^{r} p_i e_i^{\otimes 3} = p_1 e_1^{\otimes 3} + p_2 e_2^{\otimes 3}, + p_3 e_3^{\otimes 3}, \tag{3}$$

where p_i are the probabilities of these effects. Since Eq. 3 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 e_1, e_2, and 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 [2017]). Since this tensor has $O(n^3)$ entries, we can find this decomposition in $O(poly(n))$ time.

We find that the decomposition is indeed unique, following Sec. 2.3. Depending on the number of effects r we have, we construct moments of degree $d = O(\log r)$:

$$p_{\ell_{i_1},\ldots,\ell_{i_d}} := Pr[s'(\ell_{i_1}), \ldots, s'(\ell_{i_d}) | s(\neg \ell_{i_1}), \ldots, s(\neg \ell_{i_d}), a] \tag{4}$$

and the tensor decomposition we want to obtain is the following:

$$T_a = \sum_{i=1}^{r} p_i e_i^{\otimes d}, \text{where } e_i \in \{0,1\}^n, \forall i \in [r]. \tag{5}$$

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. Importantly, since this decomposition is unique, and the 0-1 vectors e_i are a feasible solution, it must be the only solution. Hence we can retrieve the effect vectors for a from T_a. The size of T_a is $n^{O(\log(r))}$, so our tensor decomposition algorithm will take $poly(n^{O(\log(r))})$ 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 given by Eq. 4 in our 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 $> \epsilon$):

$$p_{\ell_{i_1},\ldots,\ell_{i_d}} = \frac{\#\langle s,a,s'\rangle \in \mathcal{T}(a) : s'(\ell_{i_1}),\ldots,s'(\ell_{i_d}), s(\neg\ell_{i_1}),\ldots,s(\neg\ell_{i_d})}{\#\langle s,a,s'\rangle \in \mathcal{T}(a) : s(\neg\ell_{i_1}),\ldots,s(\neg\ell_{i_d})} \pm \Delta(\epsilon,\delta), \tag{6}$$

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.

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 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 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 changes 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 then as follows:

	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

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 s, if we consider the minor of the tensor T_a given by the indices of literals that are false in s, 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 $\{\ell_i,\ell_j\}$ 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. In the sections 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 $pre(a)$. Note that 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 s satisfying the precondition, the minor for the literals that are false in 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 Algorithm for Learning Stochastic Effects

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 vectors

```
input  : $O(\log(r))$-degree moments where each entry is the empirical estimate of Eq. 4,
         states $s \in \mathcal{T}(a)$.
output : global effect vectors $\{e\}$ and their probabilities $\{p\}$
1  begin
2      Reshape the moment tensor into degree-3 tensor.
3      Draw a random Gaussian vector $g$ and contract the tensor blocks $B_s$ for each $s \in \mathcal{T}(a)$ to
         matrices using $g$
4      Compute the tensor decomposition for each block and obtain 0-1 local effect vectors $\hat{e}_i$
5      while not all blocks have been reduced to zero do
6          while not all blocks have been tightened and their constraints dropped do
7              Fix a new $\lambda$: geometric search for its upper bound, then binary search for
                 tightness.
8              Solve the SDP in Eq. 7,
9              Find the tight blocks $B_s$, and obtain their eigenvectors $x_i$.
10             Un-whiten each $x_i$ to obtain $e_i$, and eliminate all $\hat{e}_{i'}$ that are inconsistent with $e_i$
                 by using Eq. 8. Drop the tight $B_s$s' constraints.
11         end
12         Combine un-whitened $e_i$ vectors into a global effect vector $e$.
13         Collect $p = \lambda_m / \langle g, e_{i_m} \rangle$ as the probability for the current global effect, where
             $\lambda_m = \min_i \lambda_i$.
14         Subtract $\lambda_m e_i^{\otimes 2}$ from their corresponding blocks.
15     end
16 end
```

Algorithm 1: Stochastic Effect Learning

of effects for each outcome. Nevertheless, we saw that we only need to produce a distribution of effects that is consistent with the moments that we can estimate. In this section, 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 vector of effects, representing a single outcome.

Our algorithm for finding a consistent decomposition to learn effects is Alg. 1. We will give a detailed discussion of the algorithm in the following subsections. 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 together, we obtain a global effect vector.

4.1 Local decomposition algorithm (Jennrich's Algorithm with whitening)

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

Firstly, pick a Gaussian random vector $g \in \mathbb{R}^n$, and compute the projection of T onto g:

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

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 process is also called contraction.

Then, we compute the singular value decomposition of T_g. Note that since $a_1, \dots, a_r$ are orthogonal, they are a set of candidate eigenvectors of T_g. Moreover, since g is Gaussian, the values $\langle g, a_i \rangle$ are going to be distinct with probability one, and hence this singular value decomposition is unique. The computed eigenvectors must be $a_1, \dots, a_r$.

In our case, $a_i = e_i$. (Note that Jennrich's algorithm requires the rank $r \leq n$, as we assume here.)

However, this requires $a_1, \dots, a_r$ to be orthogonal. In our case $a_i = e_i$. 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:

$$T_g^W = W T_g W^\top = \sum_{i=1}^{r} \langle g, e_i \rangle \cdot W e_i e_i^\top W^\top = \sum_{i=1}^{r} \langle g, e_i \rangle \cdot (W e_i)(W e_i)^\top.$$

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

4.2 Learning global outcomes by composing the local fragments

We now describe the main algorithm. As in Jennrich's Algorithm, we first sample a random Gaussian vector g to contract all the blocks, and pre-compute the tensor decomposition for each block (e.g., by Jennrich's Algorithm) and obtain their unique local tensor decomposition vectors $\{\hat{e}_i\}$. To obtain the global vector of effects, we will extract one global eigenvector at a time, then subtract it from the blocks by a common weight, similar to eigendecomposition. Since what we have is a set of contracted blocks $\{B_s\}_{s \in \mathcal{T}(a)}$, where each $B_s = (T|_{\ell:s(\neg\ell)})^W_{g|_{\ell:s(\neg\ell)}}$ (with the abuse of notation $T = T|_{\ell:s(\neg\ell)}$ referring to a local tensor block, and $g = g|_{\ell:s(\neg\ell)}$ 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. Suppose we fix a bound λ. For each coordinate i, we want to have $|(Bx)_i| \geq \lambda\, |x_i|$, where 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 x on the coordinates in the block must be an eigenvector of B.

To formulate this as a convex optimization problem (so that we can solve it efficiently), we use the Rayleigh quotient definition of the eigenvalue, written as a semidefinite program (SDP) as follows:

$$\max \lambda \; s.t. \; \langle U_s, B_s \rangle \geq \lambda, U_s \in \Lambda, \text{and } \langle U_s, U_s \rangle = 1 \text{ for all } s \in \mathcal{T}(a) \tag{7}$$

where Λ is the set of all positive semidefinite matrices. We can factorize $U = V^\top D V$ 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 some common big matrix variable U with lots of projections P_s selecting out the blocks of U, and the constraints $\langle P_s U P_s^\top, \mathbf{B} P_s U P_s^\top \rangle \geq \lambda$, where P_s selects the correct minor of U for a big block $\mathbf{B}$ that contains all the small blocks B_s.

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 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 $\{\hat{e}_{i'}\}$ that are inconsistent with the current tight vector. Then we subtract the weighted rank-1 matrix of each inconsistent $\hat{e}_{i'}$ (contracted with the same Gaussian vector) from its corresponding block B':

$$B' - w_{i'} \langle \hat{e}_{i'}, g \rangle \hat{e}_{i'}^{\otimes 2} = \sum_{i=1}^{r} w_i \langle \hat{e}_i, g \rangle \hat{e}_i^{\otimes 2} - w_{i'} \langle \hat{e}_{i'}, g \rangle \hat{e}_{i'}^{\otimes 2} = \sum_{\forall i \in [r], i \neq i'} w_i \langle \hat{e}_i, g \rangle \hat{e}_i^{\otimes 2} \tag{8}$$

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 e, and subtract $\lambda_m e_i^{\otimes 2}$ from each block, where $\lambda_m = \min_i \lambda_i$. We will collect $p = \lambda_m / \langle g, e_{i_m} \rangle$ as the probability for this global effect vector e, where $i_m = \text{argmin}_i \lambda_i$. Then we will go back to the loop of tightening all the blocks again. We will continue doing this until all the blocks are reduced to zero.

As an illustration, we return to the toy example of Figure 1. Suppose our random contraction is performed by summing 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.

Let's 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

	top	¬top	left	¬left
top	0	?	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 2: Moment tensor for updown on second iteration, after outcome "top" has been removed.

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 gives an all zero tensor and we are done. We see that we have thus obtained that the outcomes are the effect top with probability $1/2$, and the effect ¬top with probability $1/2$, which is correct.

To show the correctness of this algorithm in general, 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 $\hat{e}_{i'}$ remains consistent with e_i in each block.*

Proof. First, we note that if a post-whitening 0-1 effect vector v is not orthogonal to another post-whitening 0-1 effect vector v', then v and v' are consistent: Contrapositively, if e_i and e_j are distinct effect vectors in some block, we have chosen W so that We_i and We_j are orthogonal.

As in the analysis of Jennrich's algorithm, the eigenvalues of the contracted blocks are distinct almost surely. 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 (7) is a rank-1 matrix on that block, which is the tensor square of some vector $v = We_i$. For each other block, since the norm of the U_s matrix is 1, the U_s are convex combinations of tensor squares of vectors, where these U_s are consistent with $v^{\otimes 2}$ on any common coordinates. In particular, as above, the whitening of the corresponding projection of $e_j \neq e_i$ onto the block gives a vector orthogonal to We_i. If the tensor B_s for s only had support on components inconsistent with U_s, we would have had $\langle U_s, B \rangle = 0$, but $\langle U_s, B \rangle \geq \lambda > 0$. Therefore, for each block s, U_s must have positive weight on a rank-1 component that is the whitening of an effect vector consistent with e_i. □

Lemma 3. *The outer loop terminates after at most $r|\mathcal{T}(a)|$ iterations.*

Proof. At the end of each outer iteration, we subtract $\lambda_m x_{i_m}^{\otimes 2}$ from each block. We notice that λ_m is the eigenvalue and 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. □

Lemma 4. *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 the true value in the latent tensor decomposition $\sum_{i=1}^{r} w_i e_i^{\otimes d}$. Since the weight w_i is the probability (see Eq. 5), the local distribution is consistent. $\square$

5 Safety and Completeness

We now state our theoretical guarantees. Due to space constraints, we omit the proofs.

Theorem 1 (Safety)**.** *The probability any plan of length at most L' succeeds in Stochastic SAM action model is at most $(1+\epsilon)$ times greater than under the true model M^*. In particular, all actions that are applicable in a plan under the Stochastic SAM model are applicable to M^*.*

Note that the policy is applicable, since the precondition we produce is only stronger than the original: If $\ell \in pre(a)$ for any action a, ℓ is also a precondition for a in the learned action model.

Theorem 2 (Approximate Completeness)**.** *Fix a planner, and suppose that for the distribution $\mathcal{D}$ over problems in a domain D, the planner produces a policy that solves the problem with probability p and runs for L steps in expectation, the draw from $\mathcal{D}$, and the planner itself. Given $m \geq poly(|A|, |F|^{O(\log r)}, L, 1/\epsilon, 1/\delta)$ trajectories independently drawn from the planner on problems from $\mathcal{D}$, with probability $1-\delta$, the action model we learn satisfies the following: when a problem Π is sampled from $\mathcal{D}$ and we execute a policy of length at most L/ϵ that maximizes the probability of solving Π in the Stochastic SAM model with $L' = L/\epsilon$, Π is solved with probability at least $p - O(\epsilon)$ (over both the draw of Π and execution in M^*).*

The proofs closely follow Juba and Stern [2022]. The main difference is that the dependence on the number of fluents $|F|$ changes to the dependence on the number of moments $|F|^{O(\log r)}$.

6 Conclusions and Directions for Future Work

We have 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.

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. 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 quartic 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], an algorithm based on the power method should be much more efficient. The question is, how to handle the missing entries?

Acknowledgements

This research was supported by NSF awards IIS-1908287, IIS-1939677, and IIS-1942336. This work was partially performed while Brendan Juba was at the Simons Institute for the Theory of Computing.

References

Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins SRI, Anthony Barrett, Dave Christianson, et al. Pddl| the planning domain definition language. *Technical Report, Tech. Rep.*, 1998.

Diego Aineto, Sergio Jiménez Celorrio, and Eva Onaindia. Learning action models with minimal observability. *Artificial Intelligence*, 275:104–137, 2019.

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.

Emmanuel Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Commun. 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. *Psychometrika*, 85(1):121–153, 2020.

Stephen Cresswell and Peter Gregory. Generalised domain model acquisition from action traces. In *Proceedings of the international conference on automated planning and scheduling*, volume 21, pages 42–49, 2011.

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.

Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

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. *arXiv preprint arXiv:2203.04403*, 2022.

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.

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

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

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.

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.

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.

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

Kira Mourão, Luke Zettlemoyer, Ronald P. A. Petrick, and M ark Steedman. Learning STRIPS operators from noisy and incomplete observations. In *Proc. 28th UAI*, pages 614–623, 2012.

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.

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

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

Tselil Schramm and David Steurer. Fast and robust tensor decomposition with applications to dictionary learning. In *Conference on Learning Theory*, pages 1760–1793. PMLR, 2017.

Roni Stern and Brendan Juba. Efficient, safe, and probably approximately complete learning of action models. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 4405–4411, 2017.

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.

Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171(2-3):107–143, 2007.

Hankz Hankui Zhuo and Subbarao Kambhampati. Action-model acquisition from noisy plan traces. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.