Action Model Learning Guarantees

Brendan Juba
Washington University in St. Louis
1 Brookings Dr.
St. Louis, MO 63130 USA
bjuba@wustl.edu

Abstract

We recall a line of work on algorithms for learning action models from example trajectories that provide theoretical guarantees. These algorithms feature time and sample complexities that are polynomial in the description size of the action models, and feature the "soundness" and " approximate completeness" guarantees of the following form: any plan that is legal in the learned action model executes as predicted in the real environment, and trajectories drawn from the training distribution are legal in the learned action model with high probability. We briefly discuss the application of such algorithms to assessing the capabilities of black-box agents.

Introduction

Knowledge engineering is widely acknowledged to be extremely challenging, and this is true in the particular case of representing environments in domain modeling languages such as such as STRIPS (Fikes and Nilsson 1971), the Planning Domain Definition Language (PDDL) (Aeronautiques et al. 1998), PDDL 2.1 (Fox and Long 2003), and RDDL (Sanner 2010). Machine learning has had the primary impact of alleviating such knowledge engineering challenges, and consequently a number of methods for learning such domain models have been proposed (Cresswell and Gregory 2011; Aineto, Celorrio, and Onaindia 2019; Yang, Wu, and Jiang 2007, for example), and in particular recently in a line of work on a family of algorithms known as Safe Action Model ("SAM") learning (Stern and Juba 2017; Juba, Le, and Stern 2021; Juba and Stern 2022; Mordoch, Juba, and Stern 2023; Mordoch et al. 2023; Deng and Juba 2023; Juba, Le, and Stern 2024). The distinguishing feature of this family lies in the theoretical guarantees they provide: first, they run in polynomial time in the worst-case description size of the action models. Second, they provide a correctness guarantee of the following form: supposing that the environment is actually represented by some action model M^* from a family of action models that may be represented using at most S symbols, the learned action model $\hat{M}$ satisfies

1. *Soundness:* For any plan π formulated according to $\hat{M}$, π may be executed in M^* and executes "correctly"[1]
2. *Approximate completeness:* Suppose that there is a probability distribution $\mathcal{P}$ on planning problems (pairs of goal predicate and initial state (g, s_0)), giving rise to a probability distribution $\mathcal{D}$ on example trajectories by executing plans π^* generated by some ideal policy (planner) $\Pi^*(g, s_0)$ in the domain for (g, s_0) sampled from $\mathcal{P}$, that achieves g (reaches a state s for which g holds) from s_0 with probability p^* overall. Then given $\mathrm{poly}(S, \frac{1}{\epsilon}, \frac{1}{\delta})$ example trajectories, with probability at least $1 - \delta$ the learned action model $\hat{M}$ is such that there exists a policy $\hat{\Pi}$ that achieves a success rate in $\hat{M}$ of at least $p^* - \epsilon$ for (g, s_0) again sampled from $\mathcal{P}$.

Together, the two properties ensure that the learned model $\hat{M}$ may be passed to any domain-independent planner of the appropriate kind (i.e., lifted, stochastic, contingent, etc.) to solve most future problems that are drawn from the same distribution $\mathcal{P}$ as solved by the agent that provided the training trajectories. This framework was proposed as a "safe" approach to learning action models for two reasons. First, much as in the analogous approach to reinforcement learning known as *"Offline" Reinforcement Learning* (Kidambi et al. 2020; Yu et al. 2020; Levine et al. 2020), the method utilizes training trajectories that were provided by another agent, and does not rely on blind exploration of the environment. Here we envision a setting where a human operator provides demonstrations of how to achieve goals in the environment. Second, in most versions of the first guarantee, correct execution means that the sequence of states encountered while executing the plan π in the real environment M^* is identical to the sequence encountered during execution in the learned model $\hat{M}$. Therefore any additional safety properties may be enforced by imposing additional constraints during planning; if the plan has no violations of these constraints in the learned model, it also does not suffer them during execution in the real environment. While this does not solve the corresponding planning problem, it at least ensures that plans that are predicted to be safe solutions to the given problem are indeed safe when executed in practice. In this sense, it is a satisfactory solution to the learning component of safe execution.

AAAI 2024 Spring Symposium on User-Aligned Assessment of Adaptive AI Systems. Stanford University, Stanford, CA, USA.

[1]The precise definition of what counts as correctness varies depending on the model.

Relationship to Assessment

The SAM learning family was indeed not originally conceived of as a means to solve agent capability assessment. Nevertheless, it can be used for this task with one significant caveat: the algorithms require that the trajectories are annotated with action names and parameters (for lifted models). It turns out that under the "injective binding assumption" that is usually satisfied, the identities of the action parameters do not present a major challenge, but the identification of which actions are instances of the same operator and which are different operators is a significant assumption.

In more detail, we first recall the injective binding assumption for lifted domains. We must first recall the setting of lifted domains. For a set of objects O and a set of types T, we suppose every object $o \in O$ is associated with a type $t \in T$ denoted *type*(o). For example, in the logistics domain from the International Planning Competition (McDermott 2000) there are types *truck* and *location* and there may be objects t_1 and t_2 that represent two different trucks and two objects l_1 and l_2 that represent two different locations. A *lifted fluent* F is a pair $\langle \mathit{name}, \mathit{params} \rangle$ representing a relation over typed objects, where *name* is a symbol and *params* is a list of types. We denote the name of F and its parameters by *name*(F) and *params*(F) respectively, and $arity(F, t)$ denotes the number of type-t parameters. For example, in the logistics domain $at(?truck, ?location)$ is a lifted fluent that represents some truck $(?truck)$ is at some location $(?location)$. A *binding* of a lifted fluent F is a function $b : \mathit{params}(F) \rightarrow O$ mapping every parameter of F to an object in O of the indicated type. A *grounded fluent* f is a pair $\langle F, b \rangle$ where F is a lifted fluent and b is a binding for F. To *ground* a lifted fluent F with a binding b means to create a (Boolean-valued) fluent with a value determined by whether or not the objects in the image of b satisfy the relation associated with the lifted fluent. In our logistics example, for $F = at(?truck, ?location)$ and $b = \{?truck : truck1, ?location : loc1\}$ the corresponding grounded fluent f is $at(truck1, loc1)$, indicating whether $truck1$ is at $loc1$. The term *literal* refers to either a fluent or its negation. The definitions of binding, lifted, and grounded fluents transfer naturally to literals. Similarly, a lifted action $A \in \mathcal{A}$ is a pair $\langle \mathit{name}, \mathit{params} \rangle$ where *name* is a symbol and *params* is a list of types, denoted *name*(A) and *params*(A), respectively, and $arity(A, t)$ denotes the number of type-t parameters. Now, a *parameter binding* of a lifted literal L and an action A is a function $b_{L,A} : \mathit{params}(L) \rightarrow \mathit{params}(A)$ that maps every parameter of L to a parameter in A. A *parameter-bound literal* l for the lifted action A is a pair of the form $\langle L, b_{L,A} \rangle$ where $b_{L,A}$ is a parameter binding of L and A. In a lifted environment representation, we assume that the preconditions and effects of the action are given in terms of such parameter-bound literals. A *binding* of a lifted action A is defined like a binding of a lifted fluent, i.e., a function $b : \mathit{params}(A) \rightarrow O$. A *grounded action* a is a tuple $\langle A, b_A \rangle$ where A is a lifted action and b_A is a binding of A. For a binding of the action parameters of a lifted action A to objects in O, b_A, we then may obtain a set of ground literals $\langle L, b_A \circ b_{L,A} \rangle$ that correspond to the ground action $\langle A, b_A \rangle$. The *injective binding assumption* allows this map to be inverted. Precisely,

Definition 1 (Injective Binding) *In every grounded action $\langle A, b_A \rangle$, the binding b_A is an injective function, i.e., every parameter of A is mapped to a different object.*

If the binding is injective, there is a unique parameter binding $b_{L,A}$ satisfying $b_A \circ b_{L,A} = b_L$. Then since the effects may only include ground literals obtained through the parameter binding, the set of parameter-bound effects may be recovered from the ground effects: we simply substitute each $o \in O$ in the binding with an abstract action parameter. For example, if the actions have arity-1 of each type, this assumption is trivially satisfied. We will also need to assume that each action parameter appears in some effect fluent.

As alluded to above, the assumption that the transitions are labeled with action names is a more severe restriction. Nevertheless, given such a set of labelled trajectories corresponding to observations of an agent in some environment, we can obtain a model of the agent's capabilities as follows. The injective binding allows us to recover parameters corresponding to the bindings. We can then run a SAM algorithm on this set of trajectories to obtain a learned action model $\hat{M}$ with the guarantee that, assuming that the agent indeed can be faithfully described by some action model M^* in the corresponding class, soundness ensures that every plan $\hat{M}$ predicts is a capability of the agent is indeed a plan that the agent can execute (with the corresponding effect) and completeness ensures that with probability $1-\delta$ over the training examples, the probability that the agent exhibits a trajectory in any future episode that is not represented in $\hat{M}$ is bounded by ϵ on each episode. So, with probability $1 - \epsilon$ we could have anticipated that the exhibited trajectory is within the agent's repertoire. We note that this approach only relies on passive observations if we have the actions labelled.

The main limitation here is that the association of transitions to actions is essential to ensuring the soundness property of SAM learning. The problem is that we are learning preconditions from positive examples only (established formally in Mordoch, Juba, and Stern (2023)), so the optimal safe precondition is the set of states that satisfy all of the preconditions consistent with the states in which the action was taken in the examples. If we cannot identify more than one state as an instance of the same action, then for most representations, we obtain a precondition that only permits the action to be taken in the exact same state as presented during training, obtaining the memorized successor state. Such an algorithm does not generalize, and clearly requires a prohibitive number of examples to cover most of the agent's capabilities. But, indeed such an action model, that takes a distinct action in each transition, is consistent with the observations. So, without a further assumption we cannot generalize. It may be possible to do this by assuming for example that there is a bounded number of actions and each has some "anchoring" effect literal that distinguishes it from the rest. But, the guarantees are only as good as the assumptions.

Acknowledgements

The author was partially supported by NSF awards IIS-1908287, IIS-1939677, and IIS-1942336.

References

Aeronautiques, C.; Howe, A.; Knoblock, C.; McDermott, I. D.; Ram, A.; Veloso, M.; Weld, D.; SRI, D. W.; Barrett, A.; Christianson, D.; et al. 1998. Pddl— the planning domain definition language. *Technical Report, Tech. Rep.*

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence*, 275: 104–137.

Cresswell, S.; and Gregory, P. 2011. Generalised domain model acquisition from action traces. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 42–49.

Deng, Z.; and Juba, B. 2023. Stochastic Safe Action Model Learning. In *Seventh workshop on Generalization in Planning (GenPlan'23) at NeurIPS 2023*.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4): 189–208.

Fox, M.; and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20: 61–124.

Juba, B.; Le, H. S.; and Stern, R. 2021. Safe Learning of Lifted Action Models. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 379–389.

Juba, B.; Le, H. S.; and Stern, R. 2024. Learning Safe Action Models with Partial Observability. To appear in AAAI.

Juba, B.; and Stern, R. 2022. Learning Probably Approximately Complete and Safe Action Models for Stochastic Worlds. In *AAAI Conference on Artificial Intelligence*.

Kidambi, R.; Rajeswaran, A.; Netrapalli, P.; and Joachims, T. 2020. MOReL: Model-Based Offline Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Levine, S.; Kumar, A.; Tucker, G.; and Fu, J. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2): 13.

Mordoch, A.; Juba, B.; and Stern, R. 2023. Learning Safe Numeric Action Models. In *AAAI*, 12079–12086. AAAI Press.

Mordoch, A.; Stern, R.; Scala, E.; and Juba, B. 2023. Safe learning of PDDL domains with conditional effects. In *ICAPS'23 Workshop on Reliable Data-Driven Planning and Scheduling (RDDPS)*.

Sanner, S. 2010. Relational dynamic influence diagram language (RDDL): Language description. *Unpublished ms. Australian National University*.

Stern, R.; and Juba, B. 2017. Efficient, Safe, and Probably Approximately Complete Learning of Action Models. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 4405–4411.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2-3): 107–143.

Yu, T.; Thomas, G.; Yu, L.; Ermon, S.; Zou, J. Y.; Levine, S.; Finn, C.; and Ma, T. 2020. MOPO: Model-based Offline Policy Optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 14129–14142.