Evaluating Planning Model Learning Algorithms Across Different Representations

Anonymous Authors

Abstract

Automated planning is a prominent approach to sequential decision-making. A crucial aspect of domain-independent planning is the domain model, which provides to a planning engine the necessary application knowledge needed to synthesize solution plans. A domain model typically includes a state representation and an action model, which defines the set of possibly actions and the preconditions and effects of each action. Formulating domain-models is a challenging, time consuming, and error-prone task. For this reason, a number of approaches has been proposed to automatically learn complete (or partial) domain models from a set of provided observations. This raises the question of how to compare models learned by different approaches; there is a lack of evaluation metrics, and the complexity is exacerbated since learning algorithms may output models with different representations of states and actions. To foster the use of model learning approaches, in this paper we describe a set of metrics designed to assess different characteristics of models to be compared. Further, to bridge the potential representation gap between different learned models, we propose an encoder-decoder mechanism that allows to map two models regardless of their encoding, and we demonstrate how this encoder-decoder mechanism can be leveraged on to systematically compare models using the proposed metrics. Finally, suggest a benchmark suite based on existing domain models from the International Planning Competition (IPC) an evaluation process for using it.

1 Introduction

[Roni: TODO: What is automated planning. Focus: classical planning.]

[Roni: TODO: Creating an action model is hard. Learning action models algorithms exists.]

[Roni: TODO: Learning action models output action models that use different representations of states and actions.]

[Roni: TODO: Examples: grounded vs. lifted, different action names, different object types, paramter ordering, etc.]

[Roni: TODO: Key question: how to evaluate action model learning algorithms?] [Roni: TODO: Syntactic metrics are not enough. We need semantic metrics.]

[Roni: TODO: We propose a new paradigm for evaluating action model learning algorithms.]

[Roni: TODO: Explain high-level Encoder-decoder mechanism]

[Roni: TODO: Proposed Metrics]

[Roni: TODO: Benchmarks, evaluation methodology]

[Roni: TODO: Case study results]

2 Background and Related Work

In this section, we provide the necessary background on classical planning, and a discussion of approaches to automatically learn domain models.

2.1 Classical Planning

We focus on *classical planning* problems, which is a well-studied type of planning problem in which a single agent is acting in a fully observable, discrete, and deterministic environment. A classical planning problem is defined as a tuple $\mathcal{P} = \langle S, A, s_0, s_g \rangle$, where S is a finite set of states, A is a finite set of actions, $s_0 \in S$ is the initial state, and G is a set of goal states. An action $a \in A$ is defined as a tuple $a = \langle pre(a), \mathit{eff}(a) \rangle$, where $pre(a)$ is the precondition of a and $\mathit{eff}(a)$ is the effect of a. The precondition $pre(a)$ specifies the conditions that must hold in a state for the action a to be applicable, while the effect $\mathit{eff}(a)$ specifies the changes to the state resulting from applying the action a. In classical planning, a state is defined by a set of propositions. The precondition and effect of an action are defined as a set of literals, which are either positive or negative propositions. A solution to a classical planning is a sequence of actions that transforms the initial state s_0 into a goal state $s_g \in G$.

Planning domains are usually represented in a lifted representation, where the actions are defined with respect to a set of object types. The lifted representation of a planning domain is defined as a tuple $\mathcal{D} = \langle O, P, A \rangle$, where O is a set of object types, P is a set of predicates, and A is a set of actions. Actions and predicates are parameterized by objects, and preconditions and effects are defined accordingly. Popular classical planning systems, such FastDownward (**?**), support such a lifted representation.

Different algorithms have been proposed for learning domains for classical planning. The input to these algorithms is a set of *trajectories*. A *trajectory* is a sequence of observations and actions. An *observation* can be a state or some other information about the state, such as a set of predicates that hold in the state or a visual representation of a state. Domain model learning algorithm differ in the type of tra-

jectories they can learn from, the type of action models they can learn, and the representation of the learned action model.

2.2 Action Domain Learning Algorithms

[TODO: Open request for all: refine and add more algorithms to the following pargraphs as needed. Do not be shy: add your algorithms here.]

The ARMS algorithm (**?**) requires as observations the initial and final state of each of the provided trajectories, with optional use of intermediate states if available. For each observed action, ARMS lifts the action and constructs a set of constraints on the fluents involved in its preconditions and effects. These constraints are then resolved using a weighted MAX-SAT solver to generate the action model with the highest weight. The Simultaneous Learning and Filtering (SLAF) (**?**) learns lifted action models from trajectories even if some of the states in them are not observed. SLAF uses logical inference to filter inconsistent action models, and requires observing all actions in the trajectory. FAMA (**?**) can also handle missing observations and outputs a lifted planning domain model. It frames the task of learning an action model as a planning problem, ensuring that the returned action model is consistent with the provided observations.

NOLAM (**?**) can learn lifted action models even from noisy trajectories. LOCM (**?**) and LOCM2 (**?**), and its extension to learn action costs (**?**) analyze only the sequences of actions in the given trajectories, ignoring any information about the states between them. Based on less structured input knowledge, Framer (**?**) is an approach for learning planning domain models from natural language descriptions of activity sequences. LatPlan (**?**) and ROSAME-I (**?**) are conceptually different since they learn propositional action models from trajectories where the states in the given trajectories are given as images, as opposed to a conjunction of fluents. The **SAM!** (**SAM!**) learning algorithms (**??????**) are a family of action model learning algorithms that return action models that guarantee that plans generated with them are applicable in the actual action model. An action model having this property is referred to as *safe action model*. The **SAM!** family of algorithms addresses learning safe action models under different settings: the action models are lifted (**?**), with stochastic (**?**) or conditional effects (**?**), and even action models containing numeric preconditions and effects (**?**). **?** extended their approach to support learning safe action models in a partially observable environment. ESAM (**?**) is an extension of the **SAM!** algorithm that learns action models from domains in which there is ambiguity regarding the mapping of objects to parameters. To resolve this ambiguity, ESAM outputs a model with additional *proxy actions* that impose additional preconditions and parameter changes on actions in which such ambiguity cannot be resolved. I-ROSAME (**?**) is a recent algorithm that learns action models from trajectories where the states are given as images. They propose a neural network architecture that learns the action model in a lifted representation in tandem with a neural network that learns to map images to a given symbolic representation.

2.3 Model Reconciliation

[TODO: Pascal and Mauro will add text on model reconciliation and model merging.]

[TODO: Pascal and Mauro: can you write something about how this work relates to your model repair survey paper?]

3 Problem Setting and Objectives

In this work we consider the following domain model learning setup. An agent is acting in an environment M^*, which can be represented as a classical planning domain. The agent's actions are recorded in a set of trajectories, which are sequences of observations and actions. The observations and actions in the trajectories are given in a specific representation, which we refer to as the *input representation*. A domain model learning algorithm is given this set of trajectories, and is expected to output a domain model in classical planning. That is, this domain model can be given as input to a classical planner, and together with an appropriate problem description, the planner can generate plans with it.

[Mauro: clarify here the ground truth model, and discuss issues and alternatives.] [Roni: Note: changing *real domain model* to *environment*, to clarify that it is not a model. Then discuss what if we do have a verified domain model. TODO]

In this setup, we consider the following desired objectives of a learned domain model.

- **Syntactic similarity.** Learn a domain model that is as similar as possible syntactically the real domain model.
- **Predictive power.** Learn a domain model that allows predicting the outcome of actions in the real domain model.
- **Problem-solving ability.** Learn a domain model that enables solving problems in the real domain model.

[Mauro: reshape discussion below – align problems-solving ability to operationality notion.]

While the real domain model satisfies all three objectives, a key insight is that these objectives are in general not necessarily aligned. A learned domain model may be very *similar syntactically* to the real domain model but not effective in solving problems in it, e.g., adding redundant preconditions or missing a crucial effect. In contrast, a domain model may be very from the real domain model, e.g., adding many redundant preconditions to some actions, yet very *effective* in solving problems in the real domain since these redundant preconditions are often true in the real domain. Lastly, a domain model may be very *predictive* of behavior of the agent in the real domain model, yet too complex to be used for solving problems by a planner. For example, assume a learned domain model that has many copies of the same action in the real domain model, each with different parameters and preconditions, in order to capture different aspects of that action's behavior. While this may be useful for predicting the applicability of actions in the real domain model, it may hinder the ability of a planner to find plans for problems in the real domain with this model, due to its complexity (e.g., large branching factor).

4 Evaluation Metrics

Next, we discuss possible metrics for evaluating these three objectives. We begin by assuming that the learned domain model and the real domain model both use the same representation of states and actions, i.e., they are *syntactically similar* in the sense that they have the same action names, object types, and parameter ordering. In Section **??**, we discuss how to extend the metrics to handle cases where the learned domain model and the real domain model use different representation.

4.1 Syntactic Similarity Metrics

Prior work often focused on domain model metrics that aim for the first objective (similiarity), which are sometimes referred to as *syntactic similarly* metrics (**???**).[Roni: Others who have used, please add citations] These metrics typically compare the intersection or difference of the predicates in the actions' preconditions and effects between the learned and real domain models. We define these common metrics below. Let M and M^* be the evaluated and real domains, respectively, and let a be an action. We denote by $pre_M(a)$ the preconditions of action a according to domain M.

- True Positives: $TP_{pre}(a) = |(pre_M(a) \cap pre^*_M(a)|$
- False Positives: $FP_{pre}(a) = |((pre_M(a) \setminus pre^*_M(a)|$
- False Negatives: $FN_{pre}(a) = |(pre^*_M(a) \setminus pre_M(a))|$
- True Negatives: $TN_{pre}(a) = |L_a \setminus (pre_M(a) \cup pre^*_M(a))|$, where L_a is the set of all parameter-bound literals for an action a.

The following standard metrics from statistical analysis can then computed based on these values for each action:

- **Syntactic Precision**: $P_{pre}(a) = \frac{TP(a)}{TP(a)+FP(a)}$
- **Syntactic Recall**: $R_{pre}(a) \frac{TP(a)}{TP(a)+FN(a)}$

Other metrics such as Accuracy and F1-score can also be computed based on these values. To obtain an overall precision and recall for preconditions the entire domain model, one can compute the average of the precision and recall values for all actions: $P_{\text{avg}} = \frac{1}{|A|} \sum_{a \in A} P(a)$ and $R_{\text{avg}} = \frac{1}{|A|} \sum_{a \in A} R(a)$, where $P(a)$ and $R(a)$ are the precision and recall of action a, respectively. The same metrics can be defined for the effects of actions, with the only difference being that the literals in the effects are used instead of those in the preconditions.

[Roni: TODO: Add an example]

[Mauro: add here the notion of strong/weak equivalence, and the metrics based on graph similarity]

4.2 Predictive Power Metrics

While the syntactic similarity metrics are easy to compute, they do not accurately reflect the predictive power of the learned model. The *predictive power* metrics described next are designed to address this issue. These metrics, referred to sometimes as *semantic* domain model metrics (**???**), are based on the idea that a learned model should be able to predict the applicability of actions and their effects in the real domain model.

We define two types of predictive power metrics: *action applicability* metrics and *predicted effects* metrics. The former measures the ability of the learned model to predict whether an action is applicable in a given state, while the latter measures the ability of the learned model to predict the effects of an action in a given state. Unlike the syntactic similarity metrics, which are computed based on the action model itself, the predictive power metrics require a dataset of states that we denote by S. This dataset is intended to represent the distribution of states of interest in the domain. One way to create this dataset is by running a planner on a set of test problems with the real action model.

[Roni: TODO: Clarify this does not consider complexity of the doamin (which will be addressed in the next metric)]

Predicted applicability For a domain model M and action a, we denote by $app_M(a, S)$ the set of states in S in which a is applicable according to M. Based on this notation, we define the following predicted action applicability metric as follows for some action a:

- $\text{TP}_{app}(a) = |app_M(a, S) \cap app^*_M(a, S)|$
- $\text{FP}_{app}(a) = |app_M(a, S) \setminus app^*_M(a, S)|$
- $\text{TN}_{app}(a) = |S \setminus (app_M(a, S) \cup app^*_M(a, S))|$
- $\text{FN}_{app}(a) = |app^*_M(a, S) \setminus app_M(a, S)|$

In words, $\text{TP}_{app}(a)$ is the number of states in S where a is applicable according to both the learned model and the real model, $\text{FP}_{app}(a)$ is the number of states in S where a is applicable according to the learned model but not the real model, $\text{TN}_{app}(a)$ is the number of states in S where a is not applicable according to both models, and $\text{FN}_{app}(a)$ is the number of states in S where a is applicable according to the real model but not the learned model. From these metrics, one can compute the precision and recall of the learned model for action applicability as follows,

$$P_{app}(a) = \frac{TP_{app}(a)}{TP_{app}(a) + FP_{app}(a)} \quad (1)$$

$$R_{app}(a) = \frac{TP_{app}(a)}{TP_{app}(a) + FN_{app}(a)} \quad (2)$$

and the overall action applicability precision and recall by averaging overa all actions.

Predicted effects For domain M, action a, and state s, we denote by $a_M(s)$ the state resulting from applying a in s according to M. Based on this, we define the following predicted action effect metrics for some action a and state s, as follows:

- $\text{TP}_{eff}(s, a) = |(a_M(s) \setminus s) \cap (a^*_M(s) \setminus s)|$
- $\text{TN}_{eff}(s, a) = |s \cap a_M(s) \cap a^*_M(s)|$
- $\text{FP}_{eff}(s, a) = |(a_M(s) \setminus s) \setminus a^*_M(s)|$
- $\text{FN}_{eff}(s, a) = |(a^*_M(s) \setminus s) \setminus a^*_M(s)|$

For the purpose of the above computation, a state includes all the literals true in it, i.e., both positive propositions and negative ones. [Roni: Ensure this makes sense] One can

aggregate the above metrics over all states and actions, compute the overall values, and compute aggregated precision and recall values accordingly. The difference between the syntactic similarity of effects and the predicted effects metrics is subtle. It manifests when processing observations in which some literal ℓ is observed in the state before an action a, it is not an effect according to the learned model, but it is an effect according to the real model. This will count as a false positive in the syntactic similarity metrics yet not count as a false positive in the predicted effects metrics.

[Roni: Add a concrete example of how to compute these metrics]

4.3 Problem-Solving Metrics

[Mauro: add here disclaimer][Roni: Added the disclaimer below. Let me know what you think] Neither the syntactic similarity metrics nor the predictive power metrics are sufficient for evaluating the effectiveness of using a learned domain model to actually solve problems in the real domain. We propose two metrics that are designed to address this issue: *solving ratio* and *false plans ratio*. Both metrics are defined with respect to a set of problems P is the real domain model M^*. The solving ratio metric is defined as the ratio of problems in P that can be solved with the learned model by a given planning within a fixed limit on the available computational resources — CPU runtime and memory. We say that a problem is solved if a plan is found within the limited computational resources and that plan is valid according to the real action model. The false plans ratio metric is defined as the ratio of problems in P that are solved by the learned model but are not valid according to the real action model. This reflects the reliability of using plans with the learned model.

The straightforward nature of the above metrics is not without limitations. The ability to solve a problem with a given domain depends on external factors such as the set of test problems, the planner used, the runtime and memory budget allowed for it to run, and the computer and OS that executed the planner. Ideally, the above metrics would be run on a diverse set of test set problems, planners, resource limits, and computing machines. In practice, this might be difficult to implement but one is advised to at least run all evaluated domains on the same setup and provide an appropriate disclaimer to the concluded result.

[Roni: TODO: Add metric on the runtime of solving the problem.] [Roni: TODO: Discussion: what about a metric on how many real plans can be validated by the learned model.] [Roni: TODO: Maybe: add some unsolvability detection metrics?]

4.4 Evaluating Domain Models without a Reference Model

All the metrics above are defined based on the assumption that the ground truth model is available. However, in many cases, the ground truth model is not available. Moreover, the ground truth model may be a suboptimal representation for the actual environment.

Fortunately, with the exception of the syntactic similarity metrics, all other metrics can still be used to compare two learned model, providing statistics on which one is better according to different aspects of the environment without a ground truth model. For the predictive power metrics, we can use the same dataset of states S to compute the predicted applicability and predicted effects metrics for each evaluated model. Similarly, the problem-solving metrics can be computed for each model without a reference model, if one can execute the plans generated by the learned model in the real environment, then we can evaluate the solving ratio and false plans ratio metrics.

5 Bridging Representation Gaps

Some domain model learning algorithms output a classical planning domain that uses a different representation. Such a domain model can still be used by a planner, yet its input and output may be incompatible with that of the real domain. In some cases, these representation differences are minimal, e.g., using different ordering of the parameters or object types. In other cases, the differences are more significant. For example, the ESAM algorithm (**?**) outputs a domain model with additional *proxy actions* that are used to resolve ambiguities in the mapping of objects to parameters. The proxy actions are not part of the original domain model, and they are not used in the real domain model. [Roni: More examples?]

[Mauro: OpMaker2 generates classical planning models in OCL language **?**]

The syntactic similarity metrics are not relevant in such cases, where the representation is intentionally differetn from the real domain. Next, we describe how to apply the other metrics – predictive power and problem-solving — in such cases. Note that we limit the discussion to the case where the learned domain is still a planning domain, as oppose to images or other more involved representatios.

5.1 The Encoder-Decoder Mechanism

Let R_{M^*} denote the input representation and let R_M denote the representation of the learned domain model. The given trajectories are given in R_{M^*}, while the output of a planner that uses the learned domain model is in R_M. To allow the comparison of the learned domain model with the real domain model, we need to bridge the gap between these two representations. To this end, we require every domain model learning algorithm to output a representation *encoder* and a representation *decoder* in addition to the learned model. We describe these two components below. The *encoder* transforms states and actions in R_{M^*} to corresponding states and actions in the learned domain representation R_M. The *decoder* performs the reverse transformation, mapping states and actions in R_M to corresponding states and actions in R^*_M.

Formally, let A and A_M be the set of actions in the real and learned domain models, respectively, and let S an S_M be the set of states in the real and learned domain models, respectively. The power set is denoted by $P(X)$, which is the set of all subsets of X. The encoder is required to implement the following set of functions:

- $encodeAction : S \times A \rightarrow P(A_M)$, returns the set of actions in M that represent the application of a in a given state s.
- $encodeState : S \rightarrow S_M$, returns the state in M that represents the state s in M^*.

The decoder is required to implement the following functions:

- $decodeAction : A_M \rightarrow A$, returns the action in M^* corresponding to the action in M.
- $decodeState : S_M \rightarrow S$, returns the state in the input representation (S) that represents a given state in S_M.

5.2 Predictive Power Metrics with an Encoder-Decoder

For the predicted applicability metric, we modify the way $app_M(a, S)$ is computed. The main difference is that a single action in the real domain model may be represented by multiple actions in the learned model. Thus, we define $app_M(a, S)$ to be the number of states in S in which *there exists* an action a' in the learned model such that a' is applicable in the state in M that encodes s.

[Roni: TODO: Add a formal definition of the app_M method using the encoder and decoder methods]

Similarly, for the predicted effects metric, we need to decode the state resulting from $a_M(s)$ instead of using it as-is.

[Roni: TODO: Add a formal definition of the app_M method using the encoder and decoder methods]

5.3 Problem-Solving Metrics with an Encoder-Decoder

Adapting the problem-solving metrics to use the encoder-decoder mechanism is straightforward. The encoder is used to encode the problem from the input representation to the learned model representation. Then, the planner is run on the encoded problem with the learned model. Finally, the decoder is used to decode the solution plan from the learned model representation to the input representation. This allows checking if a plan has been found and if it is valid according to the real domain model.

5.4 Case Studies

Next, we describe how the encoder-decoder mechanism can be implemented for several action model learning algorithms.

SAM: Encoder: identity Decoder: identity

FAMA - Leonardo L.?: Encoder: identity Decoder: identity - maybe change to parameter names?

ESAM: Encoder: identity for both state and actions Decoder: translate proxy actions to original actions

ROSAME: Encoder: encode to a numeric vector of ones and zeros Decoder: map action parameters

OBSERVER: Encoder: from binding to grounded Decoder: from grounded predicates and actions to the binding they represent

[Roni: TODO: help me with the encoder-decoder for the other algorithms.]

5.5 An Evaluation Paradigm

Using the above metrics and the encoder-decoder mechanism, we can define an evaluation paradigm for action model learning algorithms. The evaluation paradigm consists of the following steps: First, we generate a set of trajectories in the input representation for learning. This set of trajectories can be generated by running a planner on a set of problems in the real domain model or via random walk. Then, this set of trajectories is split in 80/20 ratio, using 80% of the trajectories for training and 20% for testing. The evaluated learning algorithm is given the training set of trajectories and is required to output a learned domain model, an encoder, and a decoder. Following we use the learned model, encoder, and decoder to compute the metrics described above. This 80/20 split is repeated k times following a standard k-fold validation process.

6 Benchmarks

[TODO: Leonardo is in charge of this section.]

[TODO: Describe the benchmark suite and how to use it.]

[TODO: If we have time: show results for at least some of the algorithms on the benchmark suite.]

7 Discussion

The proposed evaluation paradigm is designed to be flexible and extensible. In particular, it can be extended to support other types of representations, such as images or other types of data. Another possible extension is to allow the action model learning algorithm to output a domain model that uses a more general type of planning. For example, the action model learning algorithm may output a domain model that uses a probabilistic **?** or partially observable representation **?**. Adapting the predictive power metrics to such cases may be not trivial, but the problem-solving metrics can be adapted in a straightforward manner.

Going beyond discrete representations is also possible. The predicted applicability metrics can be computed as-is, considering both numeric and discrete preconditions. More involved is the predicted effects metric, which may require defining some loss function for the numeric effects, as learning exactly the same numeric effects as in the real domain seems unlikely. Note that in all of these variants and generalizations, the encoder-decoder mechanism allows seemless use of the problem-solving metrics.

[TODO: Maybe talk about metrics for online learning] The metrics and evaluation paradigm outlined in this paper are for *offline* learning of action models, where the learning algorithm is given a set of trajectories and is required to output a domain model. *Online learning* of action models have also been studied in the literature (**?**)[TODO: More citations?], where the learning algorithm is required to learn a domain model by actively interacting with the environment. Similar to other online tasks, one may distinguish between the *cummulative regret* of the learning process, which can be the number of actions performed for learning or the number of problems failed to solve while collecting observations. Specific metrics and evaluation for online learning of action models, however is beyond the scope of this work.

8 Conclusion Future Work

We presented a new paradigm for evaluating action model learning algorithms across different representations. In this paradigm, each action model learning algorithm is required to output an action model and an encoder-decoder pair. The encoder-decoder pair is used to bridge representation gaps and enable measuring the problem-solving capabilities of the learned action models. We proposed an evaluation scheme that leverages the encoder-decoder pair to systematically compare learned action models, and described several evaluation metrics. A benchmark suite was also provided to facilitate the evaluation of action model learning algorithms, based on existing domain models from the International Planning Competition (IPC). We demonstrated our evaluation paradigm by applying it to several action model learning algorithms, including SAM, ESAM, FAMA, and ROSAME.

References

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

Amir, E., and Chang, A. 2008. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research* 33:349–402.

Asai, M., and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proceedings of the aaai conference on artificial intelligence*, volume 32, 6094–6101.

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

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using locm. *The Knowledge Engineering Review* 28(2):195–213.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Juba, B., and Stern, R. 2022. Learning probably approximately complete and safe action models for stochastic worlds. In *AAAI*, 9795–9804.

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.

Lamanna, L., and Serafini, L. 2024. Action model learning from noisy traces: a probabilistic approach. In *ICAPS*, 342–350. AAAI Press.

Le, H. S.; Juba, B.; and Stern, R. 2024. Learning safe action models with partial observability. In *AAAI Conference on Artificial Intelligence*, volume 38, 20159–20167.

Mordoch, A.; Stern, R.; Scala, E.; and Juba, B. 2023. Safe learning of pddl domains with conditional effects. In *Reliable Data-Driven Planning and Scheduling (RDDP) workshop in ICAPS*.

Mordoch, A.; Scala, E.; Stern, R.; and Juba, B. 2024. Safe learning of pddl domains with conditional effects. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 387–395.

Mordoch, A.; Juba, B.; and Stern, R. 2023. Learning safe numeric action models. In *AAAI Conference on Artificial Intelligence*, 12079–12086.

Stern, R., and Juba, B. 2017. Efficient, safe, and probably approximately complete learning of action models. In *the International Joint Conference on Artificial Intelligence (IJCAI)*, 4405–4411.

Xi, K.; Gould, S.; and Thiébaux, S. 2024. Neuro-symbolic learning of lifted action models from visual traces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 653–662.

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.