Action Model Learning based on Grammar Induction – Dissertation Abstract

Maxence Grand

Under the Supervision of Damien Pellier and Humbert Fiorino
Univ. Grenoble Alpes, LIG
3800 Grenoble, France
{Maxence.Grand, Damien.Pellier, Humbert.Fiorino}@univ-grenoble-alpes.fr

Abstract

This paper presents a novel approach to learn PDDL domain called AMLSI (*Action Model Learning with State machine Interaction*) based on grammar induction. AMLSI learns with no prior knowledge from a training dataset made up of action sequences built by random walks and by observing state transitions. The domain learnt is accurate enough to be used without human proofreading in a planner even with very highly partial and noisy observations. Thus AMLSI takles a key issue for domain learning that is the ability to plan with the learned domains. It often happens that small learning errors lead to domains that are unusable for planning. AMLSI contribution is to learn domains from partial and noisy observations with sufficient accuracy to allow planners to solve new problems. Also, this paper presents an incremental and a temporal extension.

Introduction

Hand-coding Planning domains is generally considered difficult, tedious and error-prone. The reason is that experts in charge modelling domains are not always PDDL experts and vice versa. To overcome this issue, two main approaches have been proposed. One is to develop knowledge engineering tools facilitating PDDL writing. These tools provide support for consistency and syntactic error checking, domain visualisation etc. An inconvenient aspect of these tools is that they require PDDL expertise and background in software engineering (Shah et al. 2013).

The other approach is to develop machine learning algorithms to automatically generate Pla domains as, for instance, ARMS (Yang, Wu, and Jiang 2007), SLAF (Shahaf and Amir 2006), LSONIO (Mourão et al. 2012), LOCM (Cresswell, McCluskey, and West 2013). These algorithms use training datasets made of possibly noisy and partial state/action sequences generated by a planner, or randomly generated. Classically, IPC benchmarks are used to generate training datasets. The performance of these algorithms is then measured as the syntactical distance between the learned domains and the IPC benchmarks. Machine learning approaches have three main drawbacks.

First, most of these approaches require a lot of data to perform the learning of Planning domains and in many real world applications, acquiring training datasets is difficult and costly, e.g., Mars Exploration Rover operations (Bresina et al. 2005). Also, dataset acquisition is a long term evolutive process: in real world applications, training data become available gradually over time. Moreover, in practice, it is important to be able to update learned PDDL domains to new incoming data without restarting the learning process from scratch.

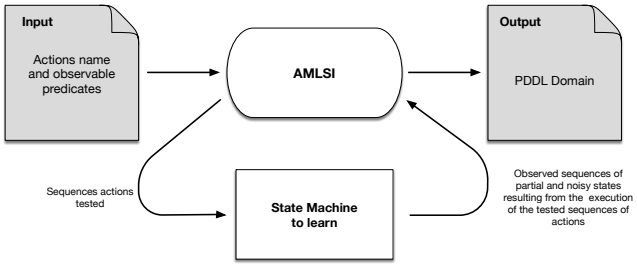


Figure 1: AMLSI: Action Model Learning with State machine Interaction

Second, the learned domains are not *accurate* enough to be used "as is" in a planner: a step of expert proofreading is still necessary to correct them. Even small syntactical errors can make sometime the learned domains useless for planning. Therefore, we consider that domain *accuracy*, that we define as the capacity of a learned domain to solve planning problems that were not used in the training dataset, is a better performance indicator than syntactical distance in practice.

Third, even if some approaches, e.g., (Mourão et al. 2012; Segura-Muros, Pérez, and Fernández-Olivares 2018; Rodrigues, Gérard, and Rouveirol 2010) are able to learn from noisy and/or partially observable data, few approaches are able to handle very high levels of noise and partial observations as can be encountered in real world applications, especially in robotics where observations are extracted using miscalibrated or noisy sensors.

To address these challenges, we propose a novel Planning domain learning algorithm called AMLSI (*Action Model Learning with State machine Interaction*). A key idea in AMLSI is that real world can be modelled as state machines and that retro-engineering real world state machines is analogous to learning a Deterministic Finite Automaton (DFA), which is equivalent to a regular grammar: we argue that (1) it is possible to learn a DFA by querying a real world

state machine (see Figure 1), and (2) to induce a PDDL domain from this regular grammar. AMLSI does not require any prior knowledge regarding the feasibility of actions in a given state, and state observations can be partial and noisy. AMLSI is highly accurate even with highly partial and noisy state observations. Thus it minimizes PDDL proofreading and correction for domain experts. Our experiments show that in many cases AMLSI does not even require any correction of the learned domains. AMLSI is lean and efficient on data consumption. It uses a supervised learning approach based on grammar induction. Training data are action/state (possibly partial and noisy) sequences labeled as feasible or infeasible. Both, feasible and infeasible action/state sequences are used by AMLSI to learn PDDL domains, thus maximizing data usability.

The rest of the paper is organized as follows. First of all, we propose some related works on PDDL domain learning. Then we present our contributions andand finally we give our Futur works.

Related Works

Many approaches have been proposed to learn Planning domains (see Table - 1). These approaches can be classified according to the input data of the learning process. The input data can be plan "traces" obtained by resolving a set of planning problems, annoted plans, decomposition tree or random walks. The input data can contain in addition to the tasks also states which can be fully observable (FO), partially observable (PO), no observable (NO), or noisy. Also, these approaches can be classified according to the output. The out can be PDDL domains and more precisely STRIPS and/or ADL domains, Temporal domains and HTN domains. In the rest of this section, we classify approaches according to the output.

PDDL Learning

A first group of approaches takes as input a set of plan traces and a partial domain, and tries to incrementally refine this domain to complete it, as for instance, OpMaker (McCluskey, Richardson, and Simpson 2002) or RIM (Zhuo, Nguyen, and Kambhampati 2013). In all of these approaches, it is assumed that the observations are complete and noiseless except OPMaker that induces operators with user interactions by using the GIPO tools (Simpson, Kitchin, and McCluskey 2007). A second group of approaches only takes plan traces as input. Most of them deals with partial observations. Among these approaches are ARMS (Yang, Wu, and Jiang 2007), LAMP (Zhuo et al. 2010), Louga (Kucera and Barták 2018), Plan-Milner algorithm (Segura-Muros, Pérez, and Fernández-Olivares 2018), AIA (Verma, Marpally, and Srivastava 2021) or FAMA (Aineto, Celorrio, and Onaindia 2019). Among these works, the ARMS systems is the most known. It gathers knowledge on the statistical distribution of frequent sets of actions in the plan traces. Then, it forms a weighted propositional satisfiability problem (weighted SAT) and solves it with a weighted MAX-SAT solver. Unlike ARMS, SLAF is able to learn actions with conditional effects. To that end, SLAF relies on building logical constraint formula based on a direct acyclic graph representation. Louga takes also as input plan traces and work with partial noiseless observations. However, Louga is able to learn actions with static properties and negative preconditions. Louga uses a genetic algorithm to learn action effects and an ad-hoc algorithm to learn action preconditions. FAMA takes as input partial plan traces, i.e. plan traces where some actions are missing, and observations are partial. This algorithm turns the task of learning into a planning problem: the learning problem is translated into a planning problem, and it resolves it by using a classical planner. Plan-Milner uses a classification algorithm based on inductive rule learning techniques: it learns action models with discrete numerical values from partial and noisy observations. The AIA algorithm learns planning domains by using a rudimentary query system. It generates plans and queries a black-box AI system with these plans to test them and updates its action model from the black-box responses. Finally, the LOCM family of action model learning approaches (Cresswell, McCluskey, and West 2013; Gregory and Cresswell 2015) works without information about initial, intermediate and final states. These algorithms extract, from plan traces, parameterized automata representing the behaviour of each object of the planning problems. Then preconditions and effects are generated from these automata. The last group of approaches takes as input random walks, that is, sets of action sequences randomly generated. Random walk approaches like IRALe (Rodrigues, Gérard, and Rouveirol 2010) deal with complete but noisy observations. IRALe is based on an online active algorithm to explore and to learn incrementally the action model with noisy observations. Other approaches such as LSONIO (Mourão et al. 2012) deal with both partial and noisy observations. LSONIO uses a classifier based on a kernel trick method to learn action models. It consists of two steps: (1) it learns a state transition function as a set of classifiers, and (2) it derives the action model from the parameters of the classifiers.

Temporal Learning

Temporal PDDL domains have different levels of action concurrency (Cushing et al. 2007). Some are sequential, which means that all plans can be rescheduled into a completely sequential succession of durative actions: each durative action starts after the previous durative action is terminated. Some temporal domains require other forms of action concurrency such as Single Hard Envelope (SHE) (Coles et al. 2009). SHE is a form of action concurrency where a durative action can be executed only if another durative action called the envelope extends over it.(Garrido and Jiménez 2020) have proposed an algorithm to learn temporal domains using CSP techniques. However this approach is limited to sequential temporal domains. To our best knowledge, there is no learning approach for both SHE and sequential temporal domains.

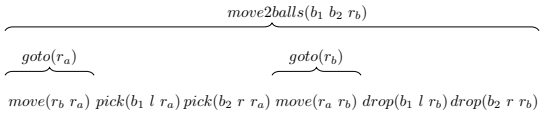
HTN Learning

First of all, the CAMEL algorithm (Ilghami et al. 2002) learns the preconditions of HTN Methods from observations of plan traces, using the version space algorithm. Then,

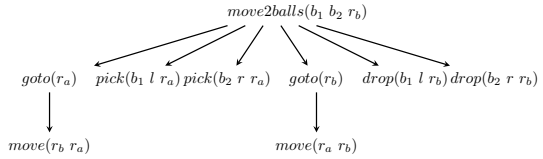
Algorithm	Input			Output			
	Input	Environment	Noise level	STRIPS	ADL	Temporal	HTN
EXPO	Partial domain, Plan traces	FO	0%	•			
RIM	Partial domain, Plan traces	FO	0%	•			
OPMaker	Partial domain, Plan traces	FO	0%	•			
ARMS	Plan traces	PO	0%	•			
Louga	Plan traces	PO	0%	•			
FAMA	Plan traces	PO	0%	•			
Plan-Milner	Plan traces	PO	10%	•			
AIA	Plan traces	FO	0%	•			
LOCM	Plan traces	NO	0%	•			
LOP	Plan traces	NO	0%	•			
NLOCM	Plan traces	NO	0%	•			
ERA	Random walk	FO	0%	•			
IRALe	Random walk	FO	20%	•			
LSONIO	Random walk	PO	5%	•			
SLAF	Plan traces	PO	0%	•	•		
LAMP	Plan traces	PO	0%	•	•		
(Garrido and Jiménez 2020)	Plans	PO	0%			•	
(Xiaoa et al. 2019)	Plans	FO	0%				•
HTN-Maker	Plans	FO	0%				•
(Hogg, Kuter, and Munoz-Avila 2010)	Plans	FO	0%				•
(Li et al. 2014)	Plans	FO	0%				•
(Garland and Lesh 2003)	Annoted Plans	FO	0%				•
CAMEL	Plans	FO	0%				•
HDL	Plans	FO	0%				•
LHTNDT	Plans	FO	0%				•
HTN-Learner	Decomposition Trees	PO	0%	•			•

Table 1: State-of-the-art action model learning algorithms. From left to right: the kind of input data, the kind of environment: **F**ully **O**bservable, **P**artially **O**bservable or **N**on **O**bservable, the maximum level of noise in observations, and and the domain expressivity STRIPS, ADL, Temporal and HTN

.



(a) An annoted plan



(b) A decomposition tree

Figure 2: HTN Learning Input examples

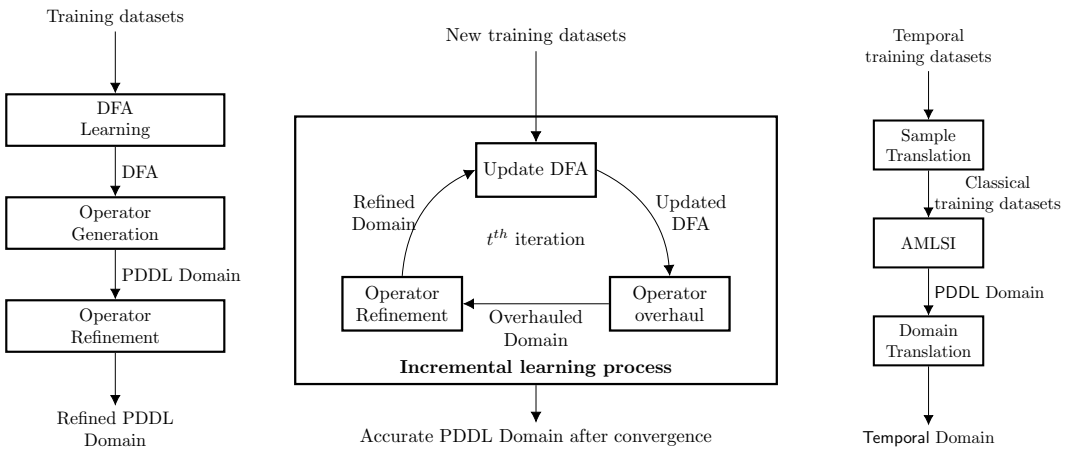
the HDL algorithm (Ilghami, Nau, and Muñoz-Avila 2006) takes as input plan traces. For each decomposition in plan traces, HDL checks if there exist a method responsible of this decomposition. If not, HDL creates a new method and initializes a new version space to capture its preconditions. Preconditions are learned in the same way as in the CAMEL algorithm. Then, (Xiaoa et al. 2019) propose an algorithm to update incomplete methods by task insertions. Then HTN-Maker (Hogg, Munoz-Avila, and Kuter 2008) takes as input plan trace generated from PDDL planner and annoted task provided by a domain expert. These approach use annoted task to build incrementally HTN Methods with preconditions. Then, (Li et al. 2014) proposed an algorithm taking as input only plan traces. This algorithm builds, from plan traces, a context free grammar (CFG) allowing to regenerate all plans. Then, methods are generated using CFG: one method for each production rule in the CFG. Then (Garland and Lesh 2003) and (Nargesian and Ghassem-Sani 2008) proposed to learn HTN Methods from annoted plan. Annoted plan (see Figure - 2a) are plan segmented with the different takes solved. Then, (Hogg, Kuter, and Munoz-Avila 2010) proposed an algorithm based on reinforcement learning. Only HTN-Learner proposes to learn Action Model and HTN Methods from decomposition trees. A decomposition tree (see Figure - 2b) is a tree corresponding to the decomposition of a method.

Contributions

In this section we will give our different contributions. More precisely, we give the method used to generate the observations. Then (see Figure - 3) we describe the algorithm used to learn PDDL domains. Then, we show how to learn PDDL domains incrementally. Finally we show how to learn Temporal domains.

Observation Generation

AMLSI produces a set of observations by using a random walk. AMLSI is able to efficiently exploit these observations by taking into account not only the fact that some actions are applicable in certain states but also that others are not. More precisely, to generate the observations, AMLSI uses random walks by applying a randomly selected action to the initial state of the problem. If this action is feasible, it is appended to the current action sequence. This procedure is repeated until the selected action is not feasible in the current state. The feasible prefix of the action sequence is then added to the set of positive samples, and the complete sequence, whose last action is not feasible, is added to the set of negative samples. This generation method allow to maximize data usability in situations where obtaining training



(a) AMLSI: PDDL domain learning overview (b) IncrAMLSI: Incremental PDDL domain learning overview (c) TempAMLSI: Temporal domain learning overview

Figure 3: Contributions overview

datasets is costly and/time-consuming.

Domain Learning

PDDL Learning An overview of the AMLSI (Grand, Fiorino, and Pellier 2020b,a) algorithm is shown in Figure 3a. AMLSI is composed of three steps:

1. *DFA Learning:* AMLSI uses an alternative version of the AMLSI algorithm with the DFA learning algorithm proposed by (Grand, Fiorino, and Pellier 2020b).
2. *Operator Generation:* AMLSI generates the set of PDDL operators from the learned DFA with the observed states.
3. *Operator Refinement:* AMLSI refines the operator preconditions and effects. This refinement steps is necessary to address partial and noisy observations.

Incremental Learning An overview of the IncrAMLSI algorithm (Grand, Fiorino, and Pellier 2021a) is shown in Figure 3b. IncrAMLSI learns incrementally the PDDL domain from incoming data and does not restart from scratch when new data become available at each iteration t. More precisely, IncrAMLSI consists in incrementally updating the PDDL domain with the new incoming training datasets available at iteration t to produce the new domain. This algorithm is made up of three steps:

1. *Update of the DFA* with the DFA learning algorithm proposed by (Grand, Fiorino, and Pellier 2020b).
2. *Overhaul of the PDDL operators* in order to add new operators and remove preconditions and effects that are no longer compatible with positive and negative samples, and the updated DFA at iteration t.
3. *Refinement of the PDDL operators* as in AMLSI algorithm to deal with noisy and partial states in observations.

The incremental process is operated each time new training datasets are input and until convergence of the PDDL domain.

Temporal Learning Some planners, such as Crikey (Halsey, Long, and Fox 2004), solve Temporal Problems by using non-temporal planners. To that end, they convert Temporal Problems into PDDL planning problems, solve them with a non-temporal planner. Then they convert the classical plan into a temporal plan with rescheduling techniques. TempAMLSI (Grand, Fiorino, and Pellier 2021b) reuse this idea in order to learn Temporal domains: TempAMLSI learns a PDDL domain and then infer a Temporal domain. TempAMLSI (summarized in Figure - 3c) has three steps.

1. *Sample Translation:* After having generated the samples of *temporal* sequences (including both feasible and infeasible sequences), TempAMLSI translates them into non-temporal sequences.
2. *PDDL Domain Learning:* TempAMLSI uses AMLSI to learn an intermediate and classical PDDL domain.
3. *Domain Translation:* TempAMLSI translates the PDDL domain into a Temporal domain.

This algorithm is able to learn both Sequential and SHE accurate Temporal domains.

Conclusion and Futur Works

As we have seen before, we presented the AMLSI algorithm for learning PDDL domains. More precisely, the AMLSI algorithm learns STRIPS-compliant domains. We plan to extend the expressivity of the learned domains and more specifically we plan to extend AMLSI to learn ADL domains. Next, we presented two extensions: (1) an incremental extension and (2) a temporal extension. We plan to develop a hierarchical extension to learn HTN domains. Finally, the AMLSI algorithm and its extension has only been tested with IPC benchmarks, we plan to test the AMLSI algorithm and its extension with real-world dataset from several fields.

References

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

Bresina, J. L.; Jónsson, A. K.; Morris, P. H.; and Rajan, K. 2005. Activity Planning for the Mars Exploration Rovers. In *Proc of ICAPS*, 40–49.

Coles, A.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. 2009. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence*, 173(1): 1–44.

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

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is Temporal Planning Really Temporal? In *Proc of IJCAI*, 1852–1859.

Garland, A.; and Lesh, N. 2003. Learning hierarchical task models by demonstration. *MERL*.

Garrido, A.; and Jiménez, S. 2020. Learning Temporal Action Models via Constraint Programming. In *Proc of ECAI*, 2362–2369.

Grand, M.; Fiorino, H.; and Pellier, D. 2020a. AMLSI: A Novel and Accurate Action Model Learning Algorithm. In *Proc of KEPS workshop*.

Grand, M.; Fiorino, H.; and Pellier, D. 2020b. Retro-engineering state machines into PDDL domains. In *Proc of ICTAI*, 1186–1193.

Grand, M.; Fiorino, H.; and Pellier, D. 2021a. IncrAMLSI: Incremental Learning of Accurate Planning Domains from Partial and Noisy Observations. In *Proc of ICTAI*, 121–128.

Grand, M.; Fiorino, H.; and Pellier, D. 2021b. TempAMLSI : Temporal Action Model Learning based on Grammar Induction. In *Proc of KEPS workshop*.

Gregory, P.; and Cresswell, S. 2015. Domain Model Acquisition in the Presence of Static Relations in the LOP System. In *Proc of ICAPS*, 97–105.

Halsey, K.; Long, D.; and Fox, M. 2004. CRIKEY-a temporal planner looking at the integration of scheduling and planning. In *Proc of the Integrating Planning into Scheduling Workshop*, 46–52.

Hogg, C.; Kuter, U.; and Munoz-Avila, H. 2010. Learning methods to generate good plans: Integrating htn learning and reinforcement learning. In *Proc of AAAI*, volume 24.

Hogg, C.; Munoz-Avila, H.; and Kuter, U. 2008. HTN-MAKER: Learning HTNs with Minimal Additional Knowledge Engineering Required. In *Proc of AAAI*, 950–956.

Ilghami, O.; Nau, D. S.; and Muñoz-Avila, H. 2006. Learning to Do HTN Planning. In *Proc of ICAPS*, 390–393.

Ilghami, O.; Nau, D. S.; Muñoz-Avila, H.; and Aha, D. W. 2002. CaMeL: Learning Method Preconditions for HTN Planning. In *Proc of ICAPS*, 131–142.

Kucera, J.; and Barták, R. 2018. LOUGA: Learning Planning Operators Using Genetic Algorithms. In *Proc of PKAW Workshop*, 124–138.

Li, N.; Cushing, W.; Kambhampati, S.; and Yoon, S. 2014. Learning probabilistic hierarchical task networks as probabilistic context-free grammars to capture user preferences. *TIST*, 5(2): 1–32.

McCluskey, T. L.; Richardson, N. E.; and Simpson, R. M. 2002. An Interactive Method for Inducing Operator Descriptions. In *Proc of ICAPS*, 121–130.

Mourão, K.; Zettlemoyer, L. S.; Petrick, R. P. A.; and Steedman, M. 2012. Learning STRIPS Operators from Noisy and Incomplete Observations. In *Proc of UAI*, 614–623.

Nargesian, F.; and Ghassem-Sani, G. 2008. LHTNDT: Learn HTN Method Preconditions using Decision Tree. In *Proc of ICINCO-ICSO*, 60–65.

Rodrigues, C.; Gérard, P.; and Rouveirol, C. 2010. Incremental Learning of Relational Action Models in Noisy Environments. In *Proc of ICLP*, 206–213.

Segura-Muros, J. Á.; Pérez, R.; and Fernández-Olivares, J. 2018. Learning Numerical Action Models from Noisy and Partially Observable States by means of Inductive Rule Learning Techniques. In *Proc of KEPS workshop*.

Shah, M.; Chrpa, L.; Jimoh, F.; Kitchin, D.; McCluskey, T.; Parkinson, S.; and Vallati, M. 2013. Knowledge engineering tools in planning: State-of-the-art and future challenges. *Knowledge engineering for planning and scheduling*, 53: 53.

Shahaf, D.; and Amir, E. 2006. Learning Partially Observable Action Schemas. In *Proc of AAAI Conference on Artificial Intelligence*, 913–919.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning domain definition using GIPO. *Knowledge Engineering Review*, 22: 117–134.

Verma, P.; Marpally, S. R.; and Srivastava, S. 2021. Asking the Right Questions: Learning Interpretable Action Models Through Query Answering. In *Proc of AAAI Conference on Artificial Intelligence*, 12024–12033.

Xiaoa, Z.; Wan, H.; Zhuoa, H. H.; Herzigb, A.; Perrusselc, L.; and Chena, P. 2019. Learning HTN Methods with Preference from HTN Planning Instances. *Proc of HPlan workshop*, 31.

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.

Zhuo, H. H.; Nguyen, T. A.; and Kambhampati, S. 2013. Refining Incomplete Planning Domain Models Through Plan Traces. In *Proc of IJCAI*, 2451–2458.

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, 174(18): 1540–1569.