An Accurate PDDL Domain Learning Algorithm from Partial and Noisy Observations

1st Maxence Grand
Univ. Grenoble Alpes, LIG
38000 Grenoble, France
Maxence.Grand@univ-grenoble-alpes.fr

2nd Damien Pellier
Univ. Grenoble Alpes, LIG
38000 Grenoble, France
Damien.Pellier@univ-grenoble-alpes.fr

3rd Humbert Fiorino
Univ. Grenoble Alpes, LIG
38000 Grenoble, France
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 tackles 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. Compared to other approaches, AMLSI uses smaller training datasets and exploits both feasible and infeasible generated action sequences.

I. INTRODUCTION

Hand-coding PDDL\textsuperscript{1} domains is generally viewed as difficult, tedious and error-prone. The reason is that the experts of the domains to model are not always PDDL experts and vice versa. To overcome this issue machine learning algorithms have been proposed to automatically generate PDDL domains as, for instance, ARMS [1], SLAF [2], LSONIO [3], LOCM [4]. These algorithms use training datasets made of state/action sequences generated by a planner, or randomly generated (random walks). Classically, IPC benchmarks are used to generate training datasets. The performance of these algorithms is measured as the syntactical distance between the learned domains and the ones used to generate the training datasets.

These approaches have three main drawbacks. First, most of these approaches require a lot of data to perform the learning of PDDL domains and in many real world applications, acquiring training datasets is difficult and costly, e.g., Mars Exploration Rover operations [5] or fleet of Autonomous Underwater Vehicles for offshore missions [6, 7]. Second, the learned domains are not enough accurate 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., [3], [8], [9] are able to learn from noisy and/or partially observable data, few approaches are able to handle very high levels of noise and high levels of partial observations as can be encountered in real world applications, especially in robotics where observations are extracted using miscalibrated or noisy sensors.

To adress these drawbacks, we propose a novel PDDL domain learning algorithm called AMLSI (Action Model Learning with State machine Interaction). AMLSI assumes it is possible to build a training dataset made up of feasible and infeasible action sequences with a trial and error method, and to observe state transitions. Based on these feasible as well as infeasible state/action sequences, AMLSI provides a PDDL domain. 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 observations. Thus it minimizes PDDL proofreading and correction for domain experts. We show experimentally 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 sequences labeled by either (partial and noisy) state observations or "failure". Both, feasible and infeasible action sequences are used by AMLSI to learn PDDL domains, thus maximizing data usability. Finally, we show that AMLSI is the only approach to our best knowledge able to learn accurate domain with such a level of partial and noisy observations (see Table I).

The rest of the paper is organized as follows. Section II proposes the problem statement. Section III details the AMLSI algorithm and section IV presents the comparative evaluation of AMLSI with state-of-the-art performance indicators on 8 IPC benchmarks.

II. PROBLEM STATEMENT

In this paper, we address STRIPS\textsuperscript{2} language in PDDL. A STRIPS planning problem is a tuple $P = (L, A, S, s_0, G, δ, τ, λ)$, where $L$ is a set of logical propositions describing the world states, $S$ is a set of state labels, $s_0 \in S$

\textsuperscript{1}Planning Domain Declaration Language

\textsuperscript{2}STanford Research Institute Problem Solver
### Table I: State-of-the-art action model learning algorithms.

| Algorithm | Input | Environment | Noise level | Accurate |
|-----------|-------|-------------|-------------|----------|
| EXPO [10] | Partial domain | FOM | 0% | • |
| RIM [11]  | Partial domain | FOM | 0% | • |
| OP/Maker [12] | Partial domain | FOM | 0% | • |
| Observer [13] | Plan traces | FOM | 0% | • |
| OPLM [14]  | Plan traces | FOM | 0% | • |
| ARMS [1]   | Plan traces | FOM | 0% | • |
| Longa [15] | Plan traces | FOM | 0% | • |
| FAMA [16]  | Plan traces | FOM | 0% | • |
| Plan-Miner [8] | Plan traces | FOM | 10% | • |
| AIA [17]   | Plan traces | FOM | 0% | • |
| LOC [4]    | Plan traces | NO | 0% | • |
| RELE [9]   | Random walk | FOM | 20% | • |
| LSONO [3]  | Random walk | FOM | 5% | • |
| AMLSI      | Random walk | FOM | 20% | • |

### III. The AMLSI Approach

AMLSI has been devised to solve STRIPS learning problems. In practice, it computes $P$ as PDDL domain and problem files. Regarding the training dataset, AMLSI uses random walks to generate $\Omega$. AMLSI assumes that $L$, $A$, $S$, and $s_0$ are known, and the observation function $\lambda$ is possibly partial and noisy. A partial observation is a state where some logical propositions are missing, and a noisy observation is a state where the truth value of some propositions is erroneous. No knowledge of the goal states $G$ is required. Once $\Sigma$ is learned, AMLSI infers the action precondition, positive and negative effect functions in $\delta$ from the state transition function $\gamma$. Finally, the operators of the PDDL domain file are induced from $\delta$.

The AMLSI algorithm consists of 4 steps: (1) generation of the observations, (2) learning of the DFA corresponding to these observations, (3) generation of the PDDL operators, and (4) refinement of the operators to cope with noisy and partial state observations.

#### A. Observation Generation

To generate the observations in $\Omega$, 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 $I_+$, the set of positive samples, and the complete sequence, whose last action is not feasible, is added to $I_-$, the set of negative samples. Random walks are repeated until $I_+$ and $I_-$ achieve an arbitrary size.

![Fig. 1: An example of DFA with pre-states and post-states](image-url)
B. DFA Learning

The learning of the DFA $\Sigma = (S, A, \gamma)$ that models the planing problem to learn is based on regular grammar induction. Regular grammar induction is a well-defined problem [19]. Many algorithms have been proposed to solve it. Among them, RPNI³ [20] is one of the few able to learn a DFA using a set of positive and negative samples. Moreover, RPNI has all the right properties: (1) it is able to identify the class of the regular languages in the limit in polynomial time, and (2) it is locally optimal, i.e., RPNI learns the smallest automaton accepting the positive samples and rejecting the negative samples, when the samples are characteristic [21]. This last property is important to reduce the number of transitions to explore while generating the PDDL operators. In practice, RPNI takes as input $I_+$ and $I_-$ (resp. $I_+$) the positive (resp. negative) samples, i.e., the feasible (resp. infeasible) action sequences, and returns a DFA accepting all the positive samples, and rejecting all the negative ones. RPNI performs a depth first search in the lattice of the acceptor trees of $I_+$ by merging nodes while checking that the new DFA obtained rejects all the negative samples contained in $I_-$. A complete description of RPNI is available in [20].

To identify a regular grammar, we need our sample to be characteristic. It is not possible to construct such a sample a priori. However, if we have a very large sample size, then it is likely that our sample is characteristic; however, one of the conditions for our approach to be effective and to be used in practice is that it requires little training data. We are faced with a contradiction. To overcome this contradiction, we propose to extend the RPNI algorithm using a heuristic. The goal of this heuristic is to allow us to reduce the amount of negative examples. We propose to force the DFA learned to accept only the sequences defined in $I_+$ and reject all other unobserved sequences. We compute all unobserved pairwise sequences (PS) $(a_i, a_j)$ and add them to the set of negative examples. The computation of the set of PS consists in computing all the possible couples of actions from the actions $A$ of the DFA and to subtract the couples present in $I_+$.

C. Operator Generation

In this step, we present how to generate $\delta = (\text{prec, add, del})$ from the learned DFA. Operator generation is based on three steps:

**Precondition generation:** To learn the preconditions $\text{prec}(a)$ of action $a$, AMLS1 computes the logical propositions that are in all the states preceding $a$ in $\Sigma$: $\text{prec}(a) = \cap s \in \text{postset}(a) \lambda(s)$

**Effect generation:** To learn the positive effects $\text{add}(a)$ of action $a$, AMLS1 computes the logical propositions that are never in states before the execution of $a$, and always present after a execution: $\text{add}(a) = \cap s \in \text{postset}(a) \lambda(s) \setminus \text{prec}(a)$

Symetrically, $\text{del}(a) = \text{prec}(a) \setminus \cap s \in \text{postset}(a) \lambda(s)$

³Regular Positive and Negative Inferences

**Action generalisation:** Once preconditions and effects are learned, actions are lifted to PDDL operators with OI-subsumption (subsumption under Object Identity) [23]. First of all, constant symbols in preconditions and effects are substituted by variable symbols. Then, the less general preconditions and effects, i.e. preconditions and effects encoding as many propositions as possible, are computed as intersection sets. This generalization method allows to ensure that all the necessary preconditions, i.e. the preconditions allowing to differentiate the states where actions are applicable from states where they are not, to be rightfully coded in the corresponding operators.

D. Operator refinement

As we assume partial and noisy observations, it is necessary to refine the PDDL operators. Operator refinement is composed of three steps.

**Effect refinement:** This step ensures that the generated operators allow to regenerate the induced regular grammar (see section III-B). We use the pre-set $\text{postset}$ to verify that for each couple of consecutive actions $a$ and $a'$ in the DFA, the effects of action $a$ applied in the state $s$ satisfy the preconditions of action $a'$. If it is not the case, we add in the effects of $a$ the propositions satisfying the preconditions of $a'$.

**Precondition refinement:** In this step, we assume like [1] that the propositions of the negative effects must be in the action preconditions. Thus for each negative effect in an operator, we add the corresponding proposition in the preconditions. Since effect refinements depend on preconditions and precondition refinements depend on effects, we repeat these two steps until convergence, i.e. no more precondition or effect is added.

**Tabu Search:** The refinement step previously described is able to find most of the preconditions and the effects of the operators even with partial observations. However, this refinement does not prevent to remove relevant or add irrelevant preconditions and effects when observations are noisy. To deal with this problem, we propose to use Tabu Search [24]. Tabu Search is a classical meta-heuristic search method employing local search methods used for mathematical optimization. The idea is to explore variants of the operators set learned in the previous step by adding and removing preconditions and effects. At each steps only variants improving the set of learned operators are kept until a local maximum is found. To determine whether one variant is better than another it is necessary to define an evaluation function. This evaluation function is called a fitness function. In practice, a variant $\Delta$ of an operators set is better than an other one given a positive and a negative set of observations if: (1) $\Delta$ accepts more positive observations, (2) $\Delta$ rejects more negative observations and (3) the state sequences produced by applying the transition function $\gamma_\Delta$ on the positive actions sequences observed in $I_+$ violate fewer preconditions and effects than the sequences produced with the other. Formally, the fitness function used by AMLS1 to evaluate a candidate
variant $\Delta$ given the observations sets $I_+$ and $I_-$ is defined as follows:

$$f(\Delta, I_+, I_-) = \sum_{\omega \in I_+} accept(\Delta, w) + \sum_{\omega \in I_-} reject(\Delta, w) + \sum_{w \in I_+} \sum_{s \in \Delta(s_0, \omega)} |s \cap \lambda(s)| - |s \setminus \lambda(s)|$$

where: $accept(\Delta, w) = 1$ if $\gamma(\Delta, s_0, w)$ is defined and $reject(\Delta, w) = 1$ if $\gamma(\Delta, s_0, w)$ is undefined.

### IV. Experiments and Evaluation

The evaluation is carried out in two stages. First, we compare the performance of AMLSI and LSONIO as a function of the size of the training dataset. To the best of our knowledge, LSONIO is the only algorithm taking as input random walks and dealing with partial and noisy observations like AMLSI. Second, we perform an ablation study to highlight the performance gain of each component of the AMLSI algorithm compared to LSONIO on 4 scenarios: (1) complete observations (100%) and no noise (0%), (2) complete observations (100%) and high level of noise (20%), (3) partial observations (25%) and no noise (0%) and (4) partial intermediate observations (25%) and high level of noise (20%).

### A. Experimental setup

Our experiments are based on 8 IPC benchmarks. All the PDDL benchmarks are STRIPS-compliant. AMLSI learns domains from one instance. To avoid performances being biased by the initial state, AMLSI is evaluated with different instances. Also, for each instance, to avoid performances being biased by the generated observations, experiments are repeated five times. Then, we generate partial observations by randomly removing a fraction of the propositions of the states, and we generate noise by changing the value of a fraction of the observable propositions. All the tests were performed on an Ubuntu 14.04 server with a multi-core Intel Xeon CPU E5-2630 clocked at 2.30 GHz with 16GB of memory. PDDL4J library [25] was used to generate the benchmark data.

### B. Evaluation metrics

Two metrics are used for the evaluation: the syntactical error [26] that computes the distance between the original domain and the learned domain, and the accuracy [11] that measures the learned domain performance to solve new problems. Formally, the syntactical error $error(\omega)$ for an operator is the Hamming distance between the learned operator and the ground truth operator, i.e. the number of extra or missing predicates in the preconditions $prec(\omega)$, the positive effects $add(\omega)$ and the negative effects $del(\omega)$ divided by the total number of possible predicates. By extension, the syntactical error for a domain composed of a set of operator $O$ is: $E_\sigma = \frac{1}{|O|} \sum_{\omega \in O} error(\omega)$. Finally, the accuracy $Acc = \frac{N^*}{N}$ is the ratio between $N$, the number of correctly solved problems with the learned domain, and $N^*$, the total number of problems to solve. In the rest of this section the accuracy is computed over 20 problems. The problems are solved with Fast Downward v19.06 [27]. Plan validation is done with VAL [28], which is used in the IPC competitions.

### C. Comparison with LSONIO

Figure 2 shows the average performance of AMLSI and LSONIO obtained on the 8 domains of our benchmarks when varying the training dataset size. The size of the training set is indicated in number of actions. For concision, we present here only the results obtained on the most difficult scenario 4 (partial intermediate observations (25%) and high level of noise (20%)). We observe that AMLSI outperforms LSONIO whatever the size of the learning dataset in terms of accuracy or in terms of syntactical distance. We also observe that AMLSI needs very little data to obtain a relatively large accuracy (almost 70% with only a learning dataset of 200 actions) in the most difficult scenario. Now, if we look at Table II to have a view of the performance of AMLSI and LSONIO by domain and by scenario, we observe that AMLSI outperforms LSONIO whatever the level of observability and noise. To summarize, AMLSI learns more accurate domain, can deal with a higher level of noise and needs less input data than LSONIO.

### D. Ablation study

Table II compares the results obtained by three variants of AMLSI: (B) Base: DFA learning is done without PS (Pairwise Sequences, see Section III-B) and without Tabu Search, (B+PS) Base + PS: DFA learning is done with PS but without Tabu Search, and (B+PS+T) Base + PS + Tabu: DFA learning is done with PS and with Tabu Search during refinement step. The Base+PS variant is more robust to partial observations than the Base variant of AMLSI. This is due to the fact that DFA learned with PS are generally better that domain learned without PS. However, when observations are noisy, the Base+PS variant is not able to learn accurate domains. Only the Base+PS+T variant is both robust to partial and noisy observations. Our ablation study confirms that adding unobserved Pairwise Sequences improves the learning of the DFA, and makes AMLSI more robust to partial observations.
while refining the preconditions and the effects by using a Tabu Search allows AMLSI to learn accurate domains with a high level of noise.

V. CONCLUSION

We have presented AMLSI, a novel algorithm to learn PDDL domains from incomplete observations with high levels of noise. AMLSI is composed of four steps. The first step consists in building two training sets of feasible and infeasible action sequences. This maximizes data usability in situations where obtaining training datasets is costly and time-consuming. In the second step, AMLSI induces a regular grammar. The third step is the generation of the PDDL operators, and the last step refines the generated operators. Our experimental results show that AMLSI outperforms the closest approach LSONIO and is able to learn accurate PDDL domains with little training data minimizing the need for expert proofreading. Future works will focus on extending AMLSI in order to learn more expressive PDDL (e.g. temporal domains, HTN domains etc.)

REFERENCES

[1] Q. Yang, K. Wu, and Y. Jiang, “Learning action models from plan examples using weighted MAX-SAT,” Artif. Intell., vol. 171, no. 2-3, pp. 107–143, 2007.

[2] D. Shahaf and E. Amir, “Learning partially observable action schemas,” in Proc of AAAI, 2006, pp. 913–919.

[3] K. Mourão, L. S. Zettlemoyer, R. P. A. Petrick, and M. Steedman, “Learning STRIPS operators from noisy and incomplete observations,” in Proc of IJCAI, 2012, pp. 614–623.

[4] S. Cresswell, T. L. McCluskey, and M. M. West, “Acquiring planning domain models using LOCM,” Knowledge Engineering Review, vol. 28, no. 2, pp. 195–213, 2013.

[5] J. L. Bresina, A. K. Jönsson, P. H. Morris, and K. Rajan, “Activity planning for the mars exploration rovers,” in Proc of IJCAI, 2005, p. 40–49.

[6] Y. Carreno, E. Pairet, Y. Petillot, and R. P. A. Petrick, “Task allocation strategy for heterogeneous robot teams in offshore missions,” in Proc of IFAAMAS, 2020, p. 222–230.

[7] C. Lesire, G. Infantes, T. Gateau, and M. Barbier, “A distributed architecture for supervision of autonomous multi-robot missions,” Auton. Robots, vol. 40, p. 1343–1362, Oct. 2016.

[8] J. Á. Segura-Muros, R. Pérez, and J. Fernández-Olivares, “Learning numerical action models from noisy and partially observable states by means of inductive rule learning techniques,” in Proc of KEPS workshop, 2018.

[9] C. Rodrigues, P. Gérard, and C. Rouveiro, “Incremental learning of relational action models in noisy environments,” in Proc of IJCAI, 2010, pp. 206–213.

[10] Y. Gil, “Learning by experimentation: Incremental refinement of incomplete planning domains,” in Proc of IJCAI, 1994, pp. 87–95.

[11] H. H. Zhao, T. A. Nguyen, and S. Kambhampati, “Refining incomplete planning domain models through plan traces,” in Proc of IJCAI, 2013, pp. 2451–2458.

[12] T. L. McCluskey, N. E. Richardson, and R. M. Simpson, “An interactive method for inducing operator descriptions,” in Proc of ICAPS, 2002, pp. 111–130.

[13] X. Wang, “Learning by observation and practice: An incremental approach for planning operator acquisition,” in Proc of IJCAI, 1995, pp. 549–557.

[14] L. Lamanna, A. Saetti, L. Serafini, A. Gerevini, and P. Traverso, “Online learning of action models for PDDL planning,” in Proc of KEPS workshop, 2021.

[15] J. Kucera and R. Barták, “LOUGA: learning planning operators using genetic algorithms,” in Proc of PKAW Workshop, 2018, pp. 124–138.

[16] D. Ameo, S. J. Celorio, and E. Otaíndia, “Learning action models with minimal observability,” Artif. Intell., vol. 275, pp. 104–137, 2019.

[17] P. Verma, S. R. Marpally, and S. Srivastava, “Asking the right questions: Learning interpretable action models through query answering,” in Proc of AAAI, 2021, pp. 12024–12033.

[18] D. Hölér, G. Behnke, P. Bercher, and S. Biundo, “Assembling the expressivity of planning formalisms through the comparison to formal languages,” in Proc of ICAPS, 2016, p. 158–165.

[19] E. M. Gold, “Language identification in the limit,” Inf. Control., vol. 10, no. 5, pp. 447–474, 1967.

[20] J. Oncina and P. Garcia, “Inferring regular languages in polynomial update time,” in Pattern Recognition and Image Analysis: Selected Papers from the IVth Spanish Symposium. World Scientific, 1992, vol. 1, pp. 49–61.

[21] P. Dupont, “Incremental regular inference,” in Proc of ICGI, 1996, pp. 222–237.

[22] D. Angluin, “Inference of reversible languages,” J. ACM, vol. 20, no. 3, pp. 741–765, 1982. [Online]. Available: https://doi.org/10.1145/322326.322334

[23] F. Esposito, G. Semeraro, N. Fanizzi, and S. Ferilli, “Multistrategy theory revision: Induction and abduction in INTHELEX,” Machine Learning, vol. 38, no. 1-2, pp. 133–156, 2000.

[24] F. W. Glover and M. Laguna, Tabu Search. Kluwer, 1997.

[25] D. Pellier and H. Fiorino, “PDDL4J: a planning domain description library for java,” J. Exp. Theor. Artif. Intell., vol. 30, no. 1, pp. 143–176, 2018.

[26] H. H. Zhao, Q. Yang, D. H. Hu, and L. Li, “Learning complex action models with quantifiers and logical implications,” Artif. Intell., vol. 174, no. 18, pp. 1540–1569, 2010.

[27] M. Helmert, “The Fast Downward planning system,” Artif. Intell., vol. 206, pp. 191–246, 2000.

[28] R. Howe and D. Long, “Val’s progress: The automatic validation tool for pddl2. 1 used in the international planning competition,” in Proc of IPC Workshop, 2003, pp. 28–37.