TempAMLSI : Temporal Action Model Learning based on Grammar Induction

Maxence Grand, Damien Pellier, Humbert Fiorino
Univ. Grenoble Alpes, LIG
Saint Martin d'Hères, France
{Maxence.Grand, Damien.Pellier, Humbert.Fiorino}@univ-grenoble-alpes.fr}

Abstract

Hand-encoding PDDL domains is generally accepted as difficult, tedious and error-prone. The difficulty is even greater when temporal domains have to be encoded. Indeed, actions have a duration and their effects are not instantaneous. In this paper, we present TempAMLSI, an algorithm based on the AMLSI approach able to learn temporal domains. TempAMLSI is based on the classical assumption done in temporal planning that it is possible to convert a non-temporal domain into a temporal domain. TempAMLSI is the first approach able to learn temporal domain with single hard envelope and Cushing's intervals. We show experimentally that TempAMLSI is able to learn accurate temporal domains, i.e., temporal domain that can be used directly to solve new planning problem, with different forms of action concurrency.

1 Introduction

Thanks to description languages like PDDL (McDermott et al. 1998), AI planning has become more and more important in many application fields. One reason is the versatility of PDDL to represent *durative actions* (Fox and Long 2003), i.e. actions that have a duration, and whose preconditions and effects must be satisfied and applied at different times.

Temporal PDDL domains have different levels of required action concurrency (Cushing et al. 2007). Some of them are *sequential*, which means that all the plan parts containing overlapping durative actions can be rescheduled into a completely sequential succession of durative actions: each durative action starts after the previous durative action is terminated. One important property of sequential temporal domains is that they can be rewritten as classical domains, and therefore used by classical non-temporal planners. Some temporal domains require different forms of action concurrence 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*, is executed simultaneously. This is due to the fact that the enveloped durative action needs a resource, during all its execution, added at the start of the execution of the envelope and deleted at the end of the execution of the envelope. One important property of SHE temporal domains is that they cannot be sequentially rescheduled.

Hand-encoding PDDL domains is generally considered difficult, tedious and error-prone by experts, and this is even more harder with action concurrency. It is therefore essential to develop tools allowing to acquire temporal domains.

To facilitate PDDL domain acquisition, different machine learning algorithms have been proposed. First, for classical domains as for instance, ARMS (Yang, Wu, and Jiang 2007), SLAF (Shahaf and Amir 2006), Louga (Kucera and Barták 2018), LSONIO (Mourão et al. 2012), LOCM (Cresswell, McCluskey, and West 2009), IRale (Rodrigues, Gérard, and Rouveirol 2010), PlanMilner (Segura-Muros, Pérez, and Fernández-Olivares 2018). In these approaches, training data are either (possibly noisy and partial) intermediate states and plans previously generated by a planner, or randomly generated action sequences. These learning techniques are promising, but they cannot be used to learn temporal domains. (Garrido and Jiménez 2020) have proposed an algorithm to learn temporal domains using CSP techniques, however their approach is limited to sequential temporal domains. To our best knowledge, there is no learning approach for both SHE and sequential temporal domains.

Several temporal planners (Fox and Long 2002; Halsey, Long, and Fox 2004; Celorrio, Jonsson, and Palacios 2015; Furelos Blanco et al. 2018) attempt to exploit classical planning techniques for temporal planning. These planners convert a temporal domain into a classical domain, i.e. a domain containing non-durative action, generate a plan using this classical domain, and use rescheduling techniques to make the plan compatible with durative actions. Our contribution is to propose an approach exploiting the conversion of temporal domains into classical domains, initially proposed to solve temporal planning problems, for the temporal domain learning task. More precisely, in this work we assume that is possible to reduce the temporal domain learning task into the classical domain learning task.

In this paper, we present TempAMLSI, a learning algorithm for temporal domains including different levels of required action concurrency. TempAMLSI is built on AMLSI (Grand, Fiorino, and Pellier 2020a), a PDDL domain learner based on grammar induction. Like AMLSI, TempAMLSI takes as input feasible and infeasible action sequences to frame what is allowed by the targeted do-

main. More precisely, TempAMLSI consists of three steps: (1) TempAMLSI converts temporal sequences into non-temporal sequences, (2) TempAMLSI learns a classical domain containing non-durative action using AMLSI, and (3) converts it into a temporal domain containing durative actions.

The rest of the paper is organized as follows. In section 2 we present a problem statement. In section 3 we give some backgrounds on AMLSI approach and, in section 4, we detail TempAMLSI steps. Finally, section 5 evaluates the performance of TempAMLSI on IPC temporal benchmarks.

2 Problem Statement

This section introduces a formalization of planning domain learning which consisting in learning a transition function of a grounded planning domain, and in expressing it as PDDL operators.

In classical planning, world states s are modeled as sets of logical propositions, and actions change the world states. Formally, let S be the set of all the propositions modeling properties of world, and A the set of all the possible actions in this world. A *state* s is a subset of S and each action $a \in A$ is a triple of proposition sets $(\rho_a, \epsilon_a^+, \epsilon_a^-)$, where $\rho_a, \epsilon_a^+, \epsilon_a^- \subseteq S$, and $\epsilon_a^+ \cap \epsilon_a^- = \emptyset$. ρ_a are the preconditions of action a, that is, the propositions that must be in the state before the execution of action a. ϵ_a^+ and ϵ_a^- are respectively the positive (add list) and the negative (del list) effects of action a, that is, the propositions that must be added or deleted in s after the execution of the action a. Therefore, learning a classical planning domain consists in learning the deterministic state transition function $\gamma : S \times A \to S$ defined as: $\gamma(s, a) = (s \cup \epsilon_a^+) \setminus \epsilon_a^-$ where $\gamma(s, a)$ exists if a is applicable in s, i.e., if and only if $\rho_a \subseteq s$.

In temporal planning (Fox and Long 2003), states are defined as in classical planning. However, the action set A is a set of durative actions. A *durative action* a is composed of:

- d_a; the duration
- $\rho_a(s), \rho_a(e), \rho_a(o)$: preconditions of a at start, over all, and at end, respectively.
- $\epsilon_a^+(s), \epsilon_a^+(e)$: positive effects of a at start and at end, respectively.
- $\epsilon_a^-(s), \epsilon_a^-(e)$: negative effects of a at start and at end, respectively.

The semantics of durative actions is defined in terms of two discrete events $start_a$ and end_a, each of which is naturally expressed as a classical action. Starting a durative action a in state s is equivalent to applying the classical action $start_a$ in s, first verifying that ρ_{start_a} holds in s. Ending a in state s' is equivalent to applying end_a in s', first by verifying that ρ_{end_a} holds in s'. $start_a$ and end_a are defined as follows:

$$\begin{array}{lll} start_a : \rho_a(s) = \rho_{start_a} & \epsilon_a^+(s) = \epsilon_{start_a}^+ & \epsilon_a^-(s) = \epsilon_{start_a}^- \\ end_a : \rho_a(e) = \rho_{end_a} & \epsilon_a^+(e) = \epsilon_{end_a}^+ & \epsilon_a^-(e) = \epsilon_{end_a}^- \end{array}$$

This process is restricted by the duration of a, denoted d_a and the over all precondition. Event end_a has to occur exactly d_a time units after $start_a$ and the over all precondition has to hold in all states between $start_a$ and end_a. Although a has a duration, its effects apply instantaneously at the start and end of a, respectively. The preconditions $\rho_a(s)$ and $\rho_a(e)$ are also checked instantaneously, but $\rho_a(o)$ has to hold for the entire duration of a. The structure of a durative action is summarized in the Figure 1.

A *temporal action sequence* is a set of action-time pairs $\pi = \{(a_1, t_1), \ldots, (a_n, t_n)\}$. Each action-time pair $(a, t) \in \pi$ is composed of a durative action $a \in A$ and a scheduled start time t of a, and induces two events $start_a$ and end_a with associated timestamps t and $t + d_a$, respectively. Events $start_a$ (resp. end_a) is applied in the state s_t (resp. s_{t+d_a}), s_t (resp. s_{t+d_a}) being a state time-stamped with t (resp. $t + d_a$). Then, the temporal transition function γ to learn can be rewritten as: $\gamma(s, a, t) = (\gamma(s_t, start_a), \gamma(s_{t+d_a}, end_a))$. The transition function $\gamma(s, a, t)$ is defined if and only if: $\rho_a(s) \in s_t$, $\rho_a(e) \in s_{t+d_a}$ and $\forall t'$ such that $t \leq t' \leq t + d_a$ $\rho_a(o) \in s_{t'}$.

To learn the state transition function γ and to express it as a PDDL temporal domain, we assume that:

- we are able to observe temporal sequences of state/action defined recursively as follows:

$$\Gamma(s, \pi) = \begin{cases} [s] & \text{if } \pi = \emptyset \\ [s] & \text{if } \gamma(s, a_0, t_0) \text{ undef} \\ [s] + \Gamma(\gamma(s, a_0, t_0), [(a_1, t_1), .., (a_n, t_n)]) & \text{otherwise} \end{cases}$$

  where observed states s can be possibly partial. A partial state is a state where some propositions are missing.
- for all action $a = (\eta_a,\ d_a,\ \rho_a(s),\ \rho_a(e),\ \rho_a(o),\ \epsilon_a^+(s),\ \epsilon_a^+(e),\ \epsilon_a^-(s),\ \epsilon_a^-(e))$ in the sequences of state/action, η_a, the name of a is known, d_a is a known constant, and $\rho_a(s)$, $\rho_a(e)$, $\rho_a(o)$, $\epsilon_a^+(s)$, $\epsilon_a^+(e)$, $\epsilon_a^-(s)$, and $\epsilon_a^-(e)$ are unknown.
- learnt temporal domains can required different forms of action concurrency (see Figure - 2) such as *Single Hard Envelope* (SHE) (Coles et al. 2009). SHE is a form of action concurrency where the execution of a durative action a is required for the execution of a second durative action a'. Formally, a SHE is a durative action a' that adds a proposition p at start and deletes it at end while p is an over all precondition of a durative action a. Contrary to sequential temporal domains, for temporal domains containing SHE there exists temporal action sequences that cannot be sequentially rescheduled. For instance, the Match domain and the the following actions:
  - $MEND(f\ m)$ such that $(light\ m) \in \rho_{MEND(f\ m)}(o)$
  - $LIGHT(m)$ such that $(light\ m) \in \epsilon^+_{LIGHT(m)}(s)$ and $(light\ m) \in \epsilon^-_{LIGHT(m)}(e)$

  The durative action $MEND(f\ m)$ cannot start before the start of the durative action $LIGHT(m)$ and $MEND(f\ m)$ cannot end after the end of $LIGHT(m)$, so $MEND(f\ m)$ have to start after the start of $LIGHT(m)$ and $MEND(f\ m)$ have to end before the end of $LIGHT(m)$, it is therefore impossible to sequentially reschedule any temporal action sequences containing these actions. Finally, note that there is other forms

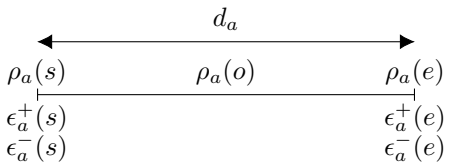

Figure 1: Structure of a durative action a

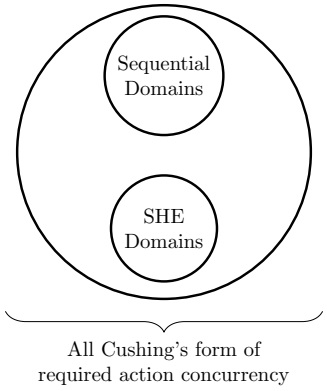

Figure 2: Different form of required action concurrency

of required action concurrency than SHE (Cushing et al. 2007).

3 Background on AMLSI

AMLSI takes as inputs I_+ and I_- training datasets, and outputs a PDDL domain. The AMLSI algorithm consists of 3 steps: (1) AMLSI learns an Deterministic Finite Automaton (DFA) corresponding to the regular grammar generating the action sequences in I_+ and forbidding those in I_-; (2) AMLSI induces the PDDL operators from the learnt DFA; (3) finally, AMLSI refines these operators to deal with noisy and partial state observations.

The first step consisting in learning the DFA is carried out by using a variant of the classic algorithm for learning regular grammar called RPNI (Oncina and García 1992). The RPNI algorithm used by AMLSI is specially tuned to deal with planning features and encode the links between preconditions and effects of actions to speedup the learning process (see (Grand, Fiorino, and Pellier 2020b) for a complete description). Formally, the learnt DFA is a quintuple (A, N, n_0, γ, F), where A is the set of actions, N is the set of nodes, $n_0 \in N$ is the initial node, γ is the node transition function, and $F \subseteq N$ is the set of final nodes.

The second step consists in generating the PDDL operators of the planning domain to learn. To carry out this step, AMLSI must first know which node of the DFA corresponds to which observed state. Thus, AMLSI maps the pairs "node, action" in the DFA with the pairs "state, action" of all $\pi \in I_+$ (see Figure 3) and labels the propositions of the node that represents the preconditions and the effects of the action transition in the DFA. Therefore, there are two different labels for a node: the (A)nte label μ_A and (P)ost label μ_P. $\mu_A(n, a)$ (resp. $\mu_P(n, a)$) gives the intersection of state set before (resp. after) the execution of the transition a in node n: a is an outcoming (resp. incoming) edge of n in the DFA. Once the labels computed, AMLSI induces the preconditions and effects the planning operators. The preconditions of an operator o are the set intersection of all the labels $\mu_A(n, a)$ such that a is an instance of o and a is an outgoing transitions of the node n. Formally, $p \in \rho_o$ if and only if for all a instance of o,

$$p \in \mu_A(n, a) \tag{1}$$

Then, the negative effects ϵ_o^- of an operator o are computed as the set intersection of the propositions present before the execution of all the actions a instances of o, and never after. Formally, $p \in \epsilon_o^-$ if and only if for all a instance of o:

$$p \in \mu_A(n, a) \wedge p \notin \mu_P(n, a) \tag{2}$$

Finally, the positive effects ϵ_o^+ of an operator o are computed similarly: $p \in \epsilon_o^+$ if and only if for all a instance of o:

$$p \notin \mu_A(n, a) \wedge p \in \mu_P(n, a) \tag{3}$$

Finally, the last step consists in refining the PDDL operators induced at step 2 to deal with noisy and partial state observations. First of all, AMLSI starts by refining the operator effects to ensure that the generated operators allow to regenerate the induced DFA. To that end, AMLSI adds all effects allowing to ensure that each transition in the automaton are feasible. Then, AMLSI refines the preconditions of the operators. AMLSI makes the following assumptions as in (Yang, Wu, and Jiang 2007): The negative effects of an operator must be used in its preconditions. Thus, for each negative effect of an operator, AMLSI adds the corresponding proposition in the preconditions. Since effect refinements depend on preconditions and precondition refinements depend on effects, AMLSI repeats these two refinements steps until convergence, i.e., no more precondition or effect is added. Finally, AMLSI performs a Tabu Search to improve the PDDL operators independently of the induced DFA, on which operator generation is based. Once the Tabu Search reaches a local optimum, AMLSI repeats all the three refinement steps until convergence.

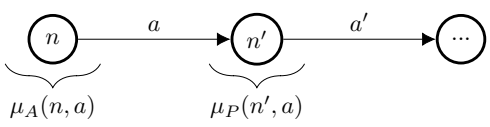

Figure 3: Mapping overview

4 Temporal AMLSI

The TempAMLSI approach is summarized by the Figure - 4. After having built the samples containing temporal sequences (including both feasible and infeasible sequences), TempAMLSI converts the temporal samples into non-temporal sequences (see Section - 4.1). We use the AMLSI algorithm to learn an intermediate classical PDDL domain, and then convert it into a temporal PDDL domain

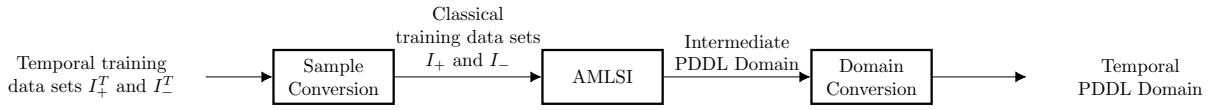

Figure 4: Overview of the TempAMLSI approach

```
(:action MEND-START
 :parameters (?f - fuse ?m - match)
 :precondition (and (handfree)
                    (light ?ma))
 :effect (and (not (handfree))))

(:action MEND-END
 :parameters (?f - f ?m - m)
 :precondition (and (light ?m))
 :effect (and (mended ?f) (handfree)))
```

(a) Classical declaration of the operator MEND - 2 Operators

```
(:action MEND-START
 :parameters (?f - fuse ?m - match)
 :precondition (and (handfree))
 :effect (and (not (handfree))))

(:action MEND-INV
 :parameters (?f - f ?m - m)
 :precondition (and (light ?m))
 :effect ())

 (:action MEND-END
 :parameters (?f - f ?m - m)
 :precondition ()
 :effect (and (mended ?f) (handfree)))
```

(b) Classical declaration of the operator MEND - 3 Operators

```
(:durative-action MEND
 :parameters (?f - fuse ?m - match)
 :duration (= ?duration 2)
 :condition (and (at start (handfree))
    (over all (light ?m)))
 :effect (and
    (at start (not (handfree)))
    (at end (mended ?f))
    (at end (handfree))
    ))
```

(c) Durative declaration of the operator MEND

Figure 5: Comparison between the durative declaration and the classical declaration of the operator MEND of the Match domain.

(see Section - 4.2). In the rest of this section we focus on the sample and domain conversion steps and present two variants for both steps:

- **2 Operators** : Actions are converted into two event actions.
- **3 Operators** : Actions are converted into three event actions.

4.1 Sample conversion

2 Operators Let us take a sample containing π^T, π^T being a positive temporal sequence such that:

$$\pi^T = \{(0, LIGHT(m)), (0.5, MEND(f_1, m)), (2.6, MEND(f_2, m))\}$$

We can convert each durative action in the following way: Each durative action a is converted into two event actions $start(a)$ and $end(a)$. After conversion, we have the following sample:

$$\pi = \{start(LIGHT(m)), start(MEND(f_1, m)), end(MEND(f_1, m)), start(MEND(f_2, m)), end(MEND(f_2, m)), end(LIGHT(m))\}$$

Negative temporal sequences are converted in the same way.

3 Operators Let us take a sample containing π^T, π^T being a positive temporal sequence such that:

$$\pi^T = \{(0, LIGHT(m)), (0.5, MEND(f_1, m)), (2.6, MEND(f_2, m))\}$$

We can convert each durative action in the following way: Each durative action a is converted into three event actions $start(a)$ $inv(a)$ and $end(a)$. After conversion, we have the following sample:

$$\pi = \{start(LIGHT(m)), inv(LIGHT(m)), start(MEND(f_1, m)), inv(LIGHT(m)), inv(MEND(f_1, m)), end(MEND(f_1, m)), inv(LIGHT(m)), start(MEND(f_2, m)), inv(LIGHT(m)), inv(LIGHT(m)), inv(MEND(f_1, m)), end(MEND(f_2, m)), end(LIGHT(m))\}$$

Negative temporal sequences are converted in the same way.

4.2 Domain conversion

After having learnt the classical PDDL domain with AMLSI, TempAMLSI converts PDDL operators into temporal operators. The Figure - 5 gives an example of conversion for the MEND operator of the Match domain for the 2 Operators and the 3 Operators variants.

2 Operators The domain conversion is done in the following way:

$$\rho_a(s) = \rho_{start(a)} \setminus \rho_{end(a)},\ \epsilon_a^+(s) = \epsilon_{start(a)}^+,\ \epsilon_a^-(s) = \epsilon_{start(a)}^-$$
$$\rho_a(e) = \rho_{end(a)} \setminus \rho_{start(a)},\ \epsilon_e^+(s) = \epsilon_{end(a)}^+,\ \epsilon_e^-(s) = \epsilon_{end(a)}^-$$
$$\rho_a(o) = \rho_{start(a)} \cap \rho_{end(a)}$$

First of all, At start (resp. at end) effects are the effects of start (resp. end) classical operators. Then, Overall preconditions are the intersection of preconditions of start and end classical operators. Finally, At start (resp. at end) preconditions are preconditions of the start (resp. end) classical operator excluding end (resp. start) preconditions.

Domain	# Operators	# Predicates	Class
Peg Solitaire	1	3	Sequential
Sokoban	2	3	Sequential
Zenotravel	5	4	Sequential
Turn and Open	5	8	SHE
Match	2	4	SHE

Table 1: Benchmark domain characteristics

3 Operators The domain conversion is done in the following way:

$$\rho_a(s) = \rho_{start(a)}\ \epsilon_a^+(s) = \epsilon_{start(a)}^+,\ \epsilon_a^-(s) = \epsilon_{start(a)}^-$$
$$\rho_a(e) = \rho_{end(a)},\ \epsilon_e^+(s) = \epsilon_{end(a)}^+,\ \epsilon_e^-(s) = \epsilon_{end(a)}^-$$
$$\rho_a(o) = \rho_{inv(a)}$$

First of all, At start (resp. at end) effects are the effects of start (resp. end) classical operators. Then, Overall preconditions are the preconditions of inv classical operators. Finally, At start (resp. at end) preconditions are preconditions of start (resp. end) classical operators.

5 Experiments and evaluations

5.1 Experimental setup

Our experiments are based on 5 temporal domains (see Table - 1). More precisely we test TempAMLSI with three Sequential domains (Peg Solitaire, Sokoban, Zenotravel), and two SHE domains (Match, Turn and Open)[1].

We deliberately choose the size of the test sets larger than the training sets to show TempAMLSI ability to learn accurate domains with small datasets. The training and test sets are generated as follows: at a given state s, we randomly choose a durative action a in A. If a is feasible, the current state is observed, and we add a to the current π. This random walk is iterated until π reaches an arbitrary length (randomly chosen between 5 and 15), and added to I_+. If a is infeasible in the current state, the concatenation of π and a is added to I_-. In the test sets E_+ and E_-, we generate action sequences with a length randomly chosen between 1 and 30.

We test each domains with three different initial states over five runs, and we use five seeds randomly generated for each run. All 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.

5.2 Evaluation Metrics

We evaluate TempAMLSI with four different metrics. The first metric, the syntactical error, is the most used metric in the literature. The other metrics, the FScore and the accuracy are more specific to our approach.

- *Syntactical Error :* The syntactical error $error(a)$ for an action a is defined as the number of extra or missing predicates in the preconditions $\rho_a(s, e, o)$, the positive effects $\epsilon_a^+(s, e)$ and the negative effects $\epsilon_a^-(s, e)$ divided by the total number of possible predicates (Zhuo et al. 2010). The syntactical error for a domain with a set of actions A is: $E_\sigma = \frac{1}{|A|}\sum_{a\in A} error(a)$.

- *FScore:* This metric is initially used for pattern recognition and binary classification (Rijsbergen 1979). Nevertheless, it can be used to evaluate the quality of a learnt grammar. Indeed, a grammar is equivalent to a binary classification system labeled with $\{1, 0\}$. For grammars we can assume that the sequences belonging to the grammar are data labeled with 1, and non-grammar sequences are data labeled with 0. This metric is therefore able to test to what extent the learnt domain D can regenerate the grammar. A domain D can regenerate a grammar if D accept, i.e. can generate, all positive test sequences $e \in E^+$ and reject, i.e. cannot generate, test negative sequences $e \in E^-$. Formally, the FScore is computed as follows: FScore $= \frac{2.P.R}{P+R}$ where R is the recall, i.e. the rate of sequences e accepted by the ground truth domain that are successfully accepted by the learnt domain, computed as follows: $R = \frac{|\{e\in E^+ \mid accept(D,e)\}|}{|E|}$ and P is the precision, i.e. the rate of sequences e accepted by the learnt domain that are sequences accepted by the ground truth domain, computed as follow: $P = \frac{|\{e\in E^+ \mid accept(D,e)\}|}{|\{e\in E^+ \mid accept(D,e)\}\cup\{e\in E^- \mid accept(D,e)\}|}$.
- *Accuracy:* It quantifies to what extent learnt domains are able to solve new planning problems (Zhuo, Nguyen, and Kambhampati 2013). Most of the works addressing the problem of learning planning domains uses the syntactical error to quantify the performance of the learning algorithm. However, domains are learnt to be used for planning, and it often happens that one missing precondition or effect, which amounts to a small syntactical error, makes them unable to solve planning problems. Formally, the accuracy $Acc = \frac{N}{N^*}$ is the ratio between N, the number of correctly solved problems with the learnt domain, and N^*, the total number of problems to solve. The accuracy is computed over 20 problems. We also report in our results the ratio of (possibly incorrectly) solved problems. In the experiments, we solve the test problems with different planners. Instances of Sequential domains and SHE domains are solved with the TP-SHE (Celorrio, Jonsson, and Palacios 2015) planner and instances of the Cushing domain are solved with the Tempo planner (Celorrio, Jonsson, and Palacios 2015). We use different planners because some planners have good results with only some forms of action concurrency. For instance, TP-SHE is the domain with the best performances for instances with Single Hard Envelope but has bad results for Cushing. Plan validation is realized with the automatic validation tool used in the IPC competition VAL (Howey and Long 2003).

5.3 Results

In order to study the performance of TempAMLSI with respect to noisy and partial state observations, we use six different experimental scenarios:

[1]Our experimental setup is available at: https://gitlab.com/AMLSI/temporal_amlsi

Domain	Observability	Noise	Algorithm	$E_\sigma(\%)$	FScore (%)	$Solved$	$Acc(\%)$
Peg	100%	0%	2 Operators	0.9%	100%	100%	100%
			3 Operators	0.9%	100%	100%	100%
		1%	2 Operators	0.9%	100%	100%	100%
			3 Operators	0.9%	100%	100%	100%
		10%	2 Operators	0.9%	100%	100%	100%
			3 Operators	1.1%	97.6%	93.3%	93.3%
	25%	0%	2 Operators	2.2%	100%	100%	100%
			3 Operators	1.4%	100%	100%	100%
		1%	2 Operators	2.2%	100%	100%	100%
			3 Operators	1.8%	94.9%	86.7%	86.7%
		10%	2 Operators	2.6%	80.3%	66.7%	86.7%
			3 Operators	1.8%	85.6%	73.3%	73.3%
Zenotravel	100%	0%	2 Operators	2.4%	100%	100%	100%
			3 Operators	2.4%	100%	100%	100%
		1%	2 Operators	2.8%	91.3%	97.7%	86.7
			3 Operators	2.7%	88.7%	99.3%	92%
		10%	2 Operators	3%	83.3%	92%	92%
			3 Operators	3.7%	56.8%	68.3%	40%
	25%	0%	2 Operators	2.4%	100%	89.7%	74.7%
			3 Operators	2.7%	100%	93%	88%
		1%	2 Operators	3.5%	71.6%	61%	52.3%
			3 Operators	3.3%	62.7%	66.7%	44.7%
		10%	2 Operators	6.3%	32.4%	58.3%	30%
			3 Operators	5.6%	40.1%	64.3%	27.7%
Sokoban	100%	0%	2 Operators	0%	100%	100%	100%
			3 Operators	0%	100%	100%	100%
		1%	2 Operators	0%	100%	100%	100%
			3 Operators	0.3%	93.6%	100%	93.3%
		10%	2 Operators	0.1%	95.1%	100%	100%
			3 Operators	0.4%	88.9%	100%	86.7%
	25%	0%	2 Operators	0.9%	100%	100%	40%
			3 Operators	0.7%	100%	100%	46.7%
		1%	2 Operators	1.9%	88.3%	86.7%	33.3%
			3 Operators	1.3%	86.7%	86.7%	33.3%
		10%	2 Operators	2%	72.3%	93.3%	46.7%
			3 Operators	1.8%	62.8%	89%	29%

Table 2: Domain learning results on 3 sequential temporal domains when observations are complete. TempAMLSI performance is measured in terms of, syntactical error E_σ, FScore and accuracy Acc.

1. Complete intermediate observations (100%) and no noise (0%).
2. Complete intermediate observations (100%) and low level of noise (1%).
3. Complete intermediate observations (100%) and high level of noise (10%).
4. Partial intermediate observations (25%) and no noise (0%).
5. Partial intermediate observations (25%) and low level of noise (1%).
6. Partial intermediate observations (25%) and high level of noise (10%).

Also, we test two different variants of TempAMLSI : (1) 2 Operators and (2) 3 Operators (see Section - 4).

Sequential Temporal Domains Table - 2 gives results for Sequential Temporal Domains.

First of all for the the Peg Solitaire domain, we observe that for the first experimental scenario (Complete observation and no noise), both variants learn optimal domains. Indeed, FScore and accuracy are optimal. However, we can observe that the syntactical distance is not optimal for both variants ($E_\sigma = 0.9\%$), this is due to the fact that some at start and at end effect preconditions are encoded as overall preconditions and vice versa. For the second experimental scenario (Complete observations and low level of noise) none of the metrics are degraded for both variants. For the third experimental scenario (Complete observations and high level of noise) the 2 Operators variant learns optimal domains. All metrics are degraded for the 3 Operators variant, however learnt domains stay accurate ($Acc = 93.3\%$). For the fourth experimental scenario (Partial observations

Domain	Observability	Noise	Algorithm	$E_\sigma(\%)$	FScore (%)	$Solved$	$Acc(\%)$
Match	100%	0%	2 Operators	3.7%	93.5%	100%	100%
			3 Operators	3.5%	93.9%	100%	100%
		1%	2 Operators	4.1%	92.8%	100%	100%
			3 Operators	4.8%	84.3%	86.7%	86.7%
		10%	2 Operators	4.3%	94.6%	86.7%	86.7%
			3 Operators	4.8%	59.5%	86.7%	86.7%
	25%	0%	2 Operators	3.8%	89.9%	93.3%	93.3%
			3 Operators	5.5%	60.3%	60%	60%
		1%	2 Operators	5.4%	80.1%	66.7%	66.7%
			3 Operators	6.7%	51%	60%	60%
		10%	2 Operators	6.5%	71.9%	73.3%	66.7%
			3 Operators	8.7%	40.6%	53.3%	53.3
Turn and Open	100%	0%	2 Operators	1.2%	100%	100%	100%
			3 Operators	1.3%	100%	100%	100%
		1%	2 Operators	1.3%	100%	100%	100%
			3 Operators	1.6%	92.9%	100%	93.3%
		10%	2 Operators	3.5%	54.1%	57%	13.3%
			3 Operators	4.8%	35%	60%	13.3%
	25%	0%	2 Operators	3.2%	100%	77%	50%
			3 Operators	2.2%	100%	97%	69.3%
		1%	2 Operators	4.1%	86.2%	62.7%	33.3%
			3 Operators	2.8%	85.6%	97%	61.7%
		10%	2 Operators	7%	41.9%	63%	20.7%
			3 Operators	6.5%	37.9%	68.3%	19.3%

Table 3: Domain learning results on 3 SHE temporal domains when observations are complete. TempAMLSI performance is measured in terms of, syntactical error E_σ, FScore and accuracy Acc.

and no noise) only the syntactical distance is degraded, FScore and Accuracy are always optimal. For the fifth experimental scenario (Complete observations and low level of noise) only the syntactical distance is degraded for the 2 Operators variant. All metrics are degraded for the 3 Operators variant, however learnt domains stay accurate ($Acc = 86.7\%$). For the last experimental scenario (Complete observations and high level of noise) all metrics are degraded for both variants. We can observe that for the majority of metrics the 3 Operators variant gives better results than the 2 Operators variant. However the 2 Operators variant gives better accuracy ($Acc = 86.7\%$) for the 2 Operators variant and $Acc = 73.3\%$ for the 3 Operators variant).

Then for the the Zenotravel domain, we observe that for the first experimental scenario, both variants learn optimal domains. For the second experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 3 Operators variant gives better results than the 2 Operators variant. For the third experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 2 Operators variant gives better results than the 3 Operators variant. We can note that only the 2 Operators variant is accurate, i.e. only the 2 Operators variant learns domains able to solve the majority of planning problems. For the fourth experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 3 Operators variant gives better results than the 2 Operators variant. For the fifth experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 2 Operators variant gives better results than the 3 Operators variant. We can note that only the 2 Operators variant is accurate. For the last experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 2 Operators variant gives better results than the 3 Operators variant. We can note that both variants are not accurate.

Finally for the the Sokoban domain, we observe that for the first experimental scenario, both variants learn optimal domains. For the second experimental scenario the 2 Operators learns optimal domains. All metrics are degraded for the 3 Operators variant, however learnt domains stay accurate. For the third experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 2 Operators variant gives better results than the 3 Operators variant. We can note that only the 2 Operators variant is accurate. For the fourth experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 3 Operators variant gives better results than the 2 Operators variant. We can note that both variants are not accurate. For the fifth experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 3 Operators variant gives better results than the 2 Operators variant. We can note that both variants are not accurate. For the last experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 2 Operators variant gives better results than the 3 Operators variant. We can note

that both variants are not accurate.

To conclude, we can observe that the 2 Operators variant is generally more robust than the 3 Operator variants. Also, for the majority of domains TempAMLSI learns accurate domains when observations are complete whatever the level of noise. When observations are partial TempAMLSI is generally not able to learn accurate domains with a high level of noise.

SHE Temporal Domains Table - 3 gives results for SHE Temporal Domains.

First of all, for the the Match domain, we observe that for the first experimental scenario, both variants does not learn optimal domains. Indeed, FScore is not optimal. This is due to the fact that learnt domains are not able to generate some feasible action sequences. However this does not affect the accuracy which is optimal for both variants. For the second experimental scenario only the syntactical distance and the FScore are degraded for the 2 Operators variant. All metrics are degraded for the 3 Operators variant, however learnt domains stay accurate. For the third experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 2 Operators variant gives better results than the 3 Operators variant. We can note that only the 2 Operators variant is accurate. For the fourth experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 3 Operators variant gives better results than the 2 Operators variant. We can note that both variants are not accurate. For the fifth experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 3 Operators variant gives better results than the 2 Operators variant. We can note that both variants are not accurate. For the last experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 3 Operators variant gives better results than the 2 Operators variant. We can note that both variants are not accurate.

Finally, for the Turn and Open domain, we observe that for the first experimental scenario, both variants learn optimal domains. For the second experimental scenario the 2 Operators variant learns optimal domains. All metrics are degraded for the 3 Operators variant, however learnt domains stay accurate. For the third experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 2 Operators variant gives better results than the 3 Operators variant. We can note that both variants are not accurate. For the fourth experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 3 Operators variant gives better results than the 2 Operators variant. For the fifth experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 3 Operators variant gives better results than the 2 Operators variant. We can note that only the 3 Operators variant is accurate. For the last experimental scenario all metrics are degraded for both variants. We can observe that for the majority of metrics the 3 Operators variant gives better results than the 2 Operators variant. We can note that both variants are not accurate.

To conclude, we can observe that the 2 Operators variant is generally more robust than the 3 Operator variants when observations are complete whatever the level of noise. When observations are partial the 2 Operators variant is more robust than the 3 Operators variant only for the match domains. Also, for all domains TempAMLSI learns accurate domains when the level of noise is not high whatever the level of observability. With a high level of noise TempAMLSI learns accurate domains only for the Match domains.

6 Conclusion

In this paper we presented TempAMLSI, a novel algorithm to learn temporal PDDL domains. TempAMLSI is based on the AMLSI approach. In this paper we reused the idea to use classical PDDL domain proposed by several temporal planners. More precisely, TempAMLSI converts the temporal sample into a sample containing non-temporal sequences, then TempAMLSI uses the AMLSI algorithm to learn a classical PDDL domain and convert it into a temporal PDDL domain, Finally, we show experimentally that the TempAMLSI approach was able to learn domains with sequential sequences and single hard envelopes with partial and noisy observations. In future works, TempAMLSI will be extended to learn temporal PDDL domain with other form of required action concurrency than Single Hard Envelopes.

Acknowledgments

This research is supported by the French National Research Agency under the "Investissements d'avenir" program (ANR-15-IDEX-02) on behalf of the Cross Disciplinary Program CIRCULAR.

References

Celorrio, S. J.; Jonsson, A.; and Palacios, H. 2015. Temporal Planning With Required Concurrency Using Classical Planning. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015*, 129–137.

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

Cresswell, S.; McCluskey, T.; and West, M. 2009. Acquisition of Object-Centred Domain Models from Planning Examples. In *Proc. of the International Conference on Automated Planning and Scheduling, ICAPS*.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is Temporal Planning Really Temporal? In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 1852–1859.

Fox, M.; and Long, D. 2002. Fast Temporal Planning in a Graphplan Framework. In *AIPS 2002 Workshop on Planning for Temporal Domains, Toulous, France, April 24, 2002*, 9–17.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.* 20: 61–124.

Furelos Blanco, D.; Jonsson, A.; Palacios Verdes, H. L.; and Jiménez, S. 2018. Forward-search temporal planning with simultaneous events. In *13th Workshop on Constraint Satisfaction Techniques for Planning and Scheduling, AAAI*.

Garrido, A.; and Jiménez, S. 2020. Learning Temporal Action Models via Constraint Programming. In *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, 2362–2369.

Grand, M.; Fiorino, H.; and Pellier, D. 2020a. AMLSI: A Novel and Accurate Action Model Learning Algorithm. In *Proc. of the International Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

Grand, M.; Fiorino, H.; and Pellier, D. 2020b. Retro-engineering state machines into PDDL domains. In *Proc. of the International Conference on Tools with Artificial Intelligence (ICTAI)*, 1186–1193.

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

Howey, R.; and Long, D. 2003. VAL's progress: The automatic validation tool for PDDL2. 1 used in the international planning competition. In *Proc. of International Planning Competition workshop (ICAPS)*, 28–37.

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

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL-the planning domain definition language.

Mourão, K.; L.Zettlemoyer; Petrick, R.; and Steedman, M. 2012. Learning STRIPS Operators from Noisy and Incomplete Observations. In *Proc. of the International Conference on Uncertainty in Artificial Intelligence*, 614–623.

Oncina, J.; and García, P. 1992. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis: Selected Papers from the IVth Spanish Symposium*, volume 1, 49–61. World Scientific.

Rijsbergen, C. 1979. *Information Retrieval*. Butterworth-Heinemann.

Rodrigues, C.; Gérard, P.; and Rouveirol, C. 2010. Incremental Learning of Relational Action Models in Noisy Environments. In *Proc. of the International Conference on Inductive Logic Programming, ILP*, 206–213.

Segura-Muros, J.; 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 International workshop on Scheduling and Knowledge Engineering for Planning and Scheduling KEPS*, 46–53.

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

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

Zhuo, H.; Nguyen, T.; and Kambhampati, S. 2013. Refining Incomplete Planning Domain Models Through Plan Traces. In *Proc. of the International Joint Conference on Artificial Intelligence, IJCAI*, 2451–2458.

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