Challenges in Evaluating Learned Planning Models in the Absence of Ground Truth Models

Antonio Garrido[1] and **Alba Gragera**[2]

[1]VRAIN, Universitat Politecnica de Valencia, Camino de Vera s/n, 46022, Valencia
[2]Computer Science and Engineering Department, Universidad Carlos III de Madrid, Spain
agarridot@dsic.upv.es, agragera@pa.uc3m.es

Abstract

Planning is the act of making a plan for something, requiring an initial state to start with and a set of goals to be achieved. A plan is formed by actions defined over an accurate action model that describes their semantics. However, the difficulty of constructing the domain of actions often presents the first bottleneck in any planning application. Approaches such as domain repair or domain learning aim to reduce the expert's effort by either repairing a given action model to align with the desired goals or, when such a model is unknown, inferring it from observations of actions/states, respectively. Resulting models are usually evaluated based on syntactic metrics with respect to the ground truth model, ignoring the fact that two syntactically different models can be semantically equivalent. But, what happens when the ground truth model is not available, which is common in the real-world? In this paper, we foster discussion about the way learned models are currently evaluated and propose possible semantic-based evaluations, focused on the dynamics of the model, and challenges arising in the absence of the ground truth model.

Introduction

Automated planning tasks typically employ the standard Planning Domain Definition Language (PDDL) (McDermott et al. 1998) to define planning problems and domains. Problems specify the initial state and goals, while domains, or action models, define available actions by detailing the conditions under which they can be applied and the effects or changes they induce in the world. Domain models must be concise but complete enough so that, when applying a sequence of actions to a given initial state it enables to reach the goals. Achieving completeness and correctness entails the primary (knowledge acquisition) bottleneck when designing action models for planning applications (Kambhampati 2007): building action models from scratch is tedious, error-prone and time-consuming. From a human perspective, designing an action model is somewhat stressful because any flaw within the domain can result in either unsolvable planning tasks or undesired plans, thus jeopardizing the usability of real-world planning. This is the main reason why there is a growing interest in automatically repairing a given action model or, ultimately, learning the model from scratch. In both cases, the idea is to acquire a better knowledge on the physics of the actions to improve the quality of the domain, while reducing the human effort in the modeling stage.

Domain Repair (DR) is the task that focuses on fixing an incomplete action model to either render an unsolvable task solvable (Gragera et al. 2023) or accommodate observed actions (Lin, Grastien, and Bercher 2023). When there is not an input action model, *Domain Learning* (DL) is the task to automatically infer the preconditions, effects, or commons structures in actions within the domain based on a set of observations (Aineto, Jiménez, and Onaindia 2019; Garrido 2023; Yang, Wu, and Jiang 2007). Several types of observations are common in the literature: plan traces with sequential or parallel actions, full or partial initial and goal states as defined in the planning problems, full or partial intermediate states, mutex (mutual exclusion) relationship information, etc.

The aim of DR and DL is to create a new action model that is *similar* to the original action model, *aka* as a *Ground Truth* (GT) or reference model. This means that the quality of the learned model is measured in terms of how similar is such a model to the GT model. How to evaluate this similarity is essential in all works that learn action models and need to assess their quality by comparison with GT. Defining a proper evaluation is not trivial and is more challenging than it seems at first sight. Traditional evaluation relies on the metrics used in pattern recognition, information retrieval, object detection and classification (Minaee et al. 2020), which perform syntactic comparisons in terms of true/false positives and negative predictions over a population of samples. This allows us to calculate different indicators such as precision, recall, F_1 score, accuracy, etc. Although these syntax-based indicators appear in almost all the works related to learning planning domains, as they are usually expected in a thorough comparison, we believe that they present some difficulties and limit the evaluation of learning models. First and most important, in order to use syntactically-oriented metrics, we assume that we have access to the GT model, which is not always the case. In most real-world problems, we have sensors that provide different types of observations, but the original action model is hidden to the observer and, consequently, unknown. Second, even when we have some clues on the GT model, it is highly unlikely to have a unique GT model; e.g., some structures of preconditions or effects could be interchangeable and depend on the original intention of the human domain designer. Therefore, a pure syntax-based evaluation might return misleading results. Loosely speaking, a

learned model can differ from a GT model but be identical to an equivalent one. Third, since both DR and DL tasks generate models that need to satisfy the observations, i.e., they are highly data-driven tasks, they might be biased toward the input data, particularly in cases where observability is partial. This often leads to action models being reformulated and compared *w.r.t.* the GT model, that are not necessarily syntactically similar, though they remain valid and capable of reproducing the dynamics of GT. In other words, the learned model might be syntactically different despite capturing analogous semantics.

The aim of this paper is to discuss the limitations of syntax-based evaluation of learned models and to establish new ways to capture the semantics of the domain based on provided input. In particular, we discuss the interest and challenges of applying a semantic evaluation, rather than a syntactic one, particularly in absence of GT models. In a semantic-oriented evaluation, the objective is to determine if the learned model has adequately captured the dynamics of the domain *w.r.t.* unknown samples.

Background

Classical Planning

A classical planning domain is typically defined as the tuple $\delta = \langle \mathcal{P}, \mathcal{A} \rangle$, where $\mathcal{P}$ is a set of Boolean predicates and $\mathcal{A}$ represents the set of actions[1] that define the action model in GT. We use the semantics introduced in STRIPS (Fikes and Nilsson 1971), where each action $a \in \mathcal{A}$ has a set of positive preconditions ($\mathsf{pre}(a)$) and a set of positive and negative effects that are asserted and retracted, respectively ($\mathsf{eff}(a) = \{\mathsf{eff}^+(a) \cup \mathsf{eff}^-(a)\}$); $\mathsf{pre}(a), \mathsf{eff}^+(a), \mathsf{eff}^-(a) \subseteq \mathcal{P}$. An action can be applied when all its preconditions hold, and the effects happen after its application.

A planning problem ρ for a domain δ is defined as the tuple $\rho = \langle \delta, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{I}$ represents a full state, which assigns a true/false value to all predicates in $\mathcal{P}$, and $\mathcal{G}$ represents a partial goal state with the true predicates in $\mathcal{P}$ to be reached. It is a partial state because it does not necessarily include all the predicates in $\mathcal{P}$.

A plan trace for ρ, or simply a plan, is a set $\pi = \{\langle t_1, a_1 \rangle, \langle t_2, a_2 \rangle \dots \langle t_n, a_n \rangle\}$. Each $\langle t_i, a_i \rangle$ contains an action a_i, which is a fully grounded version of an action $a \in \mathcal{A}$, and t_i as the time when a_i happens. Note the plan can include parallel actions. π induces a chronologically-ordered sequence of full states $\langle \mathcal{S}_0 \dots \mathcal{S}_{end} \rangle$, where $\mathcal{S}_0 = \mathcal{I}$ and $\mathcal{G} \subseteq \mathcal{S}_{end}$. The plan length is the time for the state $\mathcal{S}_{end}$.

A Simple Example

An example, which we will use through the paper, that belongs to the well-known *blocksworld* domain is shown in Figure 1. It includes predicates such as *(clear ?x)* and *(ontable ?x)* to be used as preconditions/effects in the actions *pick-up* and *stack*. Let us consider that, in this domain, we define a planning problem with five blocks {*A,R,S,T,Y*}.

[1] $\mathcal{A}$ is also known as the set of operators or lifted (i.e., non-grounded) actions.

```
(:action pick-up
 :parameters (?x - block)
 :precondition (and (clear ?x) (ontable ?x) (handempty))
 :effect (and (not (ontable ?x)) (not (clear ?x))
              (not (handempty)) (holding ?x)))

(:action stack
 :parameters (?x - block ?y - block)
 :precondition (and (holding ?x) (clear ?y))
 :effect (and (not (holding ?x)) (not (clear ?y))
              (clear ?x) (handempty) (on ?x ?y)))
```

Figure 1: Example of two actions of the blocksworld domain. The domain contains two additional actions *put-down* and *unstack*, as the inverse of *pick-up* and *stack*, respectively.

```
1: (pick-up Y)
2: (stack Y S)
3: (pick-up A)
4: (stack A Y)
5: (pick-up R)
6: (stack R A)
7: (pick-up T)
8: (stack T R)
```

Figure 2: Optimal plan of length 8 to form the stack "*TRAYS*" in the blocksworld domain. Only *pick-up* and *stack* actions are needed in the plan.

For simplicity, all blocks are initially clear and on the table: $\mathcal{I}$={*(clear T), (ontable T), (clear R), (ontable R), (clear A), (ontable A)*...}. We want tor reach the stack of blocks that forms the word "*TRAYS*": $\mathcal{G}$={*(clear T), (on T R), (on R A)*...}, i.e., *T* is on the top and *S* is on the table. The optimal plan for this problem is depicted in Figure 2.

Observations

We define a set of observations $\mathcal{O} = \{\mathcal{O}_i\}$, ranging from the observations over a single plan ($|\mathcal{O}|$=1, i.e., one-shot learning) to over hundreds or thousands of plans. In particular, each $\mathcal{O}_i$ represents a sequence of observations over the plan π_i and is defined as $\mathcal{O}_i = \langle o_{i,1}, o_{i,2} \dots o_{i,n} \rangle$. The time when each $o_{i,j}$ is observed (*time*($o_{i,j}$)) is typically unknown, but $\mathcal{O}_i$ represents a chronological sequence that preserves the ordering in π_i; that is, *time*($o_{i,j}$) $\leq$ *time*($o_{i,j+1}$).

There are two types of observations, depending on the input information that is observed over π_i:

- Action-based observation, where an observation represents an action being executed in π_i, e.g., $o_{i,j}$=*(pick-up A)*.[2]
- State-based observation, where an observation is the result of watching the true/false value of one or more predicates over the execution of π_i. The (ideal) most informative case is to observe a full state with all the pred-

[2] Although it is not an observation in itself, some extra knowledge can be included to represent the fact that two actions are mutex and cannot occur simultaneously. In other words, if two observed actions $o_{i,j}$ and $o_{i,k}$ are mutex, then *time*($o_{i,j}$) $\neq$ *time*($o_{i,k}$).

icates, e.g., $o_{i,j}$=$\mathcal{I}$, but a partial state can also be observed because sensoring all predicates at the same time is unlikely, e.g., $o_{i,j}$=$\mathcal{G}$. The simplest, least informative case, is to observe just one predicate, e.g., $o_{i,j}$=$\{$*(clear A)=true*$\}$.

While some approaches are limited to specific types of observations (e.g., only intermediate states or complete plan traces of actions), many of them are designed to accept a variable amount of input data, thus comprising a combination of: initial and goal states, action observations, mutex information, etc. (Aineto, Jiménez, and Onaindia 2019; Garrido 2022; Kucera and Barták 2018; Yang, Wu, and Jiang 2007).

Domain Repair and Domain Learning Tasks

A DR task works with a partially specified or flawed domain, aiming to modify it to enable the generation of plans that achieve the goals or to align resulting plans with given action observations. The repaired model is an *app*roximation of $\mathcal{A}$ in GT, denoted as $App(\mathcal{A})$. Typically, $App(\mathcal{A})$ is just an approximation of the actions in δ because the repaired actions are not always identical to the original ones in GT, as some preconditions or effects can be missing (incomplete) or mislearned (incorrect). In literature, $App(\mathcal{A})$ is typically compared with the reference GT model using syntactic-oriented comparisons, thus measuring the similarity between δ and $App(\mathcal{A})$. However, the quality of the reparation usually goes beyond syntactic metrics and highly depends on the provided observations. For example, let us consider a scenario where the goal is to pick up block *Y*, and the *pick-up* action lacks the effect of holding it. An action observation such as *(pick-up Y)* would aid in aligning the domain with the GT model, as it indicates the action that should perform it. In this case, the repaired model $App(\mathcal{A})$ maintains the capability to hold blocks within the domain's dynamics and the comparison with the GT model would be correct. On the contrary, in the absence of observations, modifying any domain action to add the effect of holding the block would suffice to achieve the goal. For example, any action that involves block *Y*, such as *(stack A Y)* or *(stack Y A)*, could learn the effect *(holding Y)*. In this case, $App(\mathcal{A})$ maintains again the capability to hold blocks, but the comparison with the GT model would be incorrect now.

In a DR task, the action model is already partially specified and a few observations are provided as input. In cases where the model is entirely empty, a DR task is not enough and a DL task becomes necessary to learn the model from scratch. This way, DR can be considered as a specific case of DL. The idea of a DL task is to build an action model from scratch, also denoted as an approximate model $App(\mathcal{A})$, that satisfies the highest number of the observations in $\mathcal{O}$ (ideally, all of them). For example, Figure 3 shows a potential action model learned for *pick-up* and *stack*. In *pick-up*, the preconditions are both correct and complete (all the preconditions in the GT and the learned action match entirely), whereas the effects are correct but incomplete (only the positive effect has been learned). In *stack*, the preconditions are complete (no precondition in the GT action is missing) but incorrect (the last precondition should not have been learned), whereas the effects are correct but incomplete (again, no negative effects have been learned).

```
(:action pick-up
 :parameters (?x - block)
 :precondition (and (clear ?x) (ontable ?x) (handempty))
 :effect (and (holding ?x)))

(:action stack
 :parameters (?x - block ?y - block)
 :precondition (and (holding ?x) (clear ?y) (clear ?x))
 :effect (and (clear ?x) (handempty) (on ?x ?y)))
```

Figure 3: Example of a potential action model learned for *pick-up* and *stack*. The actions are an approximation of the reference actions in Figure 1.

The learned model in $App(\mathcal{A})$ highly depends on the given observations. For example, no negative information has been observed to learn the model shown in Figure 3. Consequently, there is no need to learn negative effects. In other words, once one predicate is asserted it is never retracted, thus remaining forever true. This is the reason why *(clear ?x)* is learned as a precondition in *stack*. Intuitively, the block *?x* had to be clear at any moment before executing *stack* and, in absence of negative effects, it will maintain such a state.

Syntactic-based Evaluation Metrics

Syntactically-oriented metrics for learning action models keep the classical notation used in machine learning classification. Although these metrics are standard, we include how they are calculated to improve the readability of the paper. Let *TP, FP, TN, FN* be true positive, false positive, true negative, and false negative, respectively, in $App(\mathcal{A})$ calculated over a number *N* of preconditions and effects of the actions $\mathcal{A}$ in the GT model. For each action learned, *TP* stands for a precondition/effect that is both in $App(\mathcal{A})$ and in GT, i.e., it is a successful match, whereas a *TN* is not present in any of them. In terms of *TP* and *TN*, the precondition/effect is syntactically identical in $App(\mathcal{A})$ and in GT. *FP* stands for a precondition/effect that is learned in $App(\mathcal{A})$ but it is not in GT (it should not have been learned), whereas a *FN* is in GT but it has not been learned in $App(\mathcal{A})$. In terms of *FP* and *FN*, the precondition/effect is different in $App(\mathcal{A})$ and in GT. These positive and negative values allow us to define different performance indicators, among others:

- Missing information and error counting, defined as *FN*. Intuitively, one count of error occurs if an action precondition/effect is not learned in $App(\mathcal{A})$ when it should.
- Accuracy and error rate. Accuracy is defined as *(TP+TN)/N*, while error rate is defined as *(FP+FN)/N*. Accuracy provides relative indicators over the total number of preconditions or effects in GT, and error rate the opposite.
- Precision. It is the fraction of relevant instances among the learned ones, and provides an idea of how error-free

and sound $App(\mathcal{A})$ is. It is defined as *TP/(TP+FP)*. Precision=1 means no *FP*.

- Recall or TP rate. It is the fraction of the total amount of relevant instances that are learned, and gives an idea of how complete $App(\mathcal{A})$ is. It is defined as *TP/(TP+FN)*. Recall=1 means no *FN*. Alternatively, FN rate=1-TP rate.
- Specificity or TN rate. It is the proportion of correctly not learned instances over the instances not to be learned. It is defined as *TN/(TN+FP)*. Specificity=1 means no *FP*. Alternatively, FP rate=1-TN rate.
- F_1 score. It is the harmonic mean of the precision and recall, which analyzes equally both values in one metric. It is defined as *2·Precision·Recall/(Precision+Recall)*. In perfect learning Precision=Recall=1, which implies F_1=1.
- ROC (Receiver Operating Characteristic) curves to graphically compare TP rates *vs.* FP rates, or TN rates *vs.* FN rates, in order to evaluate the false instances that are learned. In the ROC curve, the diagonal line represents a random guess to learn and the points above/below the diagonal represent better/worse performance than such random learning.
- AUC (Area Under the Curve). It provides a summary of the ROC curve. In perfect learning AUC=1, which means $App(\mathcal{A})$ correctly distinguishes between all the positive and the negative instances. If AUC=0, all negative instances are learned as positives and all positive instances as learned as negatives.

The main limitations of these metrics are twofold. First, they ignore the fact that two syntactically different models can generate equivalent dynamics. Second, they require (and rely on) the GT model. Overcoming the first limitation while retaining the GT model may not be a significant challenge. Various alternative metrics derived from the GT can address this issue. For example, comparing whether the learned model and the GT can generate the same states/number of states, the number of the expanded nodes, the initial heuristic value or the heuristic regions generated using both domains, etc. Overcoming the second limitation is more challenging, as there is not a reference model on which to rely. In the remainder of the paper, we present various proposals and challenges for evaluating a learned domain model in the absence of the GT model, relying only on the provided input data. We categorize these proposals into scenarios characterized by different levels of observability.

Semantic-based Evaluation Metrics in Absence of GT

Once the DR/DL task has found an action model, we need to evaluate its quality. Let us revisit the simple example introduced above. The original domain δ that represents the GT model is the one depicted in Figure 1. Let us assume that the action model learned $App(\mathcal{A})$ is the one depicted in Figure 3, where the negative effects are not learned. A pure syntactical-oriented comparison between δ and $App(\mathcal{A})$ returns different performance indicators, e.g., Precision$\approx$1, FN$>$0, error rate$>$0, Recall$<$1, $F_1<1$, etc. This variability in the indicators, e.g., almost perfect result *w.r.t.* the precision but worse result *w.r.t.* the recall, is a limitation for the evaluation of the true quality of $App(\mathcal{A})$. One could be tempted to avoid this syntax-based comparison by defining new planning problems and use $App(\mathcal{A})$ to find plans that should be compared with the plans created when using δ. Although this overcomes the limitations of the syntactical-oriented comparison, it still requires an absolute knowledge of δ, as defined in GT. But what happens when GT is unknown or not entirely known? In such a case, a type of evaluation that does not depend on GT is necessary. Since we do not have access to GT, we cannot solve new planning problems either. In consequence, we need to evaluate the quality of $App(\mathcal{A})$ *w.r.t.* the observations given in $\mathcal{O}$. In order to assess whether $App(\mathcal{A})$ can reproduce unknown observations without inconsistencies, typical learning approaches split the dataset of observations into two disjoint sets and run a two-fold cross-validation evaluation. Thus, $\mathcal{O}$ is distributed into two sets: the first set for the DR/DL task and the second one for the evaluation, denoted as $\mathcal{O}_{RL}$ and $\mathcal{O}_{ev}$, respectively.

We now analyze the observations in $\mathcal{O}_{ev}$, and propose semantic-based alternatives for the evaluation of $App(\mathcal{A})$ *w.r.t.* all observations in $\mathcal{O}_{ev}$. Intuitively, the underlying idea of the semantic evaluation is to test whether $App(\mathcal{A})$ has learned the essential dynamics of GT, thanks to $\mathcal{O}_{RL}$, that satisfy $\mathcal{O}_{ev}$. We define four scenarios. The Scenarios 1, 2 and 3 are organized *w.r.t.* the observability of the observations, that is, full, partial and null observability. These three scenarios assume the observations are always noiseless, which means that observed values are actual values with no uncertainty. The Scenario 4 relaxes such assumption and allows for uncertain observations.

Scenario 1. Full observability: initial and goal states + actions in the plan + intermediate states

This is the task that uses the most complete and informative set of observations, where in every $\mathcal{O}_i \in \mathcal{O}_{ev}$ we observe the initial and goal states in ρ_i, the entire plan π_i, and all its intermediate states $S_{i,j}$; that is, $\mathcal{O}_i = \langle \mathcal{S}_{i,0} = \mathcal{I}_i, a_{i,1}, \mathcal{S}_{i,1}, a_{i,2}, \mathcal{S}_{i,2} \ldots a_{i,n}, \mathcal{S}_{i,end} \supseteq \mathcal{G}_i \rangle$. Despite having access to full observability, only positive information is observed in $\mathcal{G}_i$ and $\mathcal{S}_{i,j}$, with j$>$0, as we follow the STRIPS assumptions.[3]

Evaluation proposal and challenges. We propose to use the learned model in $App(\mathcal{A})$ as the new domain to solve the planning problem with $\mathcal{I}_i$ and $\mathcal{G}_i$, i.e., $\rho_{App} = \langle App(\mathcal{A}), \mathcal{I}_i, \mathcal{G}_i \rangle$ to find a plan π_{App} that solves it. Then, π_{App} is evaluated to check whether it fully satisfies each $\mathcal{O}_i$. This can be considered as a semantic-based evaluation. The advantage of this type of evaluation is twofold. First, it focuses on the semantics of the actions and the way they change the world states, rather than on their syntax. Second,

[3]Note that, if all the intermediate states $\mathcal{S}_{i,j}$ are full states, the learning task becomes easier as we can derive the effects of every action $a_{i,j}$ from the difference between $\mathcal{S}_{i,j}$ and $\mathcal{S}_{i,j-1}$, and we have only to learn its preconditions.

it does not need to have access to GT, as it only uses the observation $\mathcal{O}_i$.

Let us assume the observed plan π_i is the one depicted in Figure 2, where the goal $\mathcal{G}_i$ is to reach *TRAYS*, being all the blocks initially clear and on the table ($\mathcal{I}_i$). A potential plan π_{App} that solves ρ_{App} is shown in Figure 4. As can be seen, both the observed plan π_i and the new one π_{App} reach the same goals in $\mathcal{G}_i$, starting from the same $\mathcal{I}_i$, no matter the quality or optimality of π_i and π_{App}. Additionally, π_{App} induces almost the same sequence of partial states $\langle \mathcal{S}_{i,1}, \mathcal{S}_{i,2} \dots \mathcal{S}_{i,end} \rangle$ observed in $\mathcal{O}_i$.[4] This evaluation is fairer than a syntactic one: in terms of the current $\mathcal{O}_i$, $App(\mathcal{A})$ can be considered as semantically equivalent to GT because it reaches the same goals and traverses very similar states.

The fact of having access to full observability is an advantage for the semantic evaluation, because it remains valid even if: 1) the intermediate states are missing, but $\mathcal{G}_i$ is still observed; or 2) $\mathcal{G}_i$ is missing, but the intermediate states are still observed; or 3) π_i is empty (no actions are observed whatsoever) but the intermediate states or $\mathcal{G}_i$ are still observed. Consequently, this evaluation is valid for a different range of observations. In any case, the idea is that $App(\mathcal{A})$ allows us to find a plan π_{App} that reaches the observed goals or traverses the observed states induced by π_i, when available.

One may argue four important aspects regarding the semantic evaluation. First, $App(\mathcal{A})$ is an approximation of GT. In this example, the negative effects are missing in $App(\mathcal{A})$, so GT and $App(\mathcal{A})$ will never be equivalent. Although this is absolutely true, the absence of negative information in $App(\mathcal{A})$ is due to its absence in $\mathcal{O}_i$, and the task must learn a model according to the given observations. Second, the actions in the resulting plans are the same, but they are not planned in the same order; actually, π_{App} is a parallel plan, whereas π_i is clearly sequential. But it is important to note that, without negative effects, π_{App} could be sequentialized in exactly the same order as π_i (actions in a plan are temporally planned due to causal links or the particular ordering the planner decides). Third, how to deal with scenarios where there are multiple plans that solve the same goals? For example, when π_i is an optimal plan but π_{App} is not, or vice versa, or when there might be some multiple optimal plans. In these cases, there might be differences *w.r.t.* the actions and the evaluation would get low scores. Fourth, $App(\mathcal{A})$ and π_{App} are evaluated when using the observed $\mathcal{I}_i$ and $\mathcal{G}_i$ in $\mathcal{O}_i$, rather than using a brand new problem. However, if we want to compare with a brand new problem we will require to know the action model in GT, which does not make too much sense when we are trying to learn the action model. Therefore, although these four aspects are certainly convincing and represent a challenge, it is also convincing that both GT and $App(\mathcal{A})$ are somewhat equivalent and show the same utility in this first scenario.

Finally, it is important to note that a learned model

[4]The only difference in the sequence of partial states is given by the negative effects: once a predicate is achieved by π_{App} it will be maintained forever, whereas in π_i it can appear and disappear.

```
1: (pick-up T)
1: (pick-up R)
1: (pick-up A)
1: (pick-up Y)
2: (stack T R)
2: (stack R A)
2: (stack A Y)
2: (stack Y S)
```

Figure 4: A potential plan π_{App} to solve ρ_{App} in Scenario 1.

$App(\mathcal{A})$ that fails to reach some (intermediate or goal) state in $\mathcal{O}_i$ cannot be considered semantically equivalent to GT. For example, let us imagine that the *stack* action does not learn the effect *(on ?x ?y)*. In such a case, there will be no plan to solve ρ_{App}, which means that $App(\mathcal{A})$ is not an equivalent approximation to the actions in GT. In this case, $App(\mathcal{A})$ is neither syntactically nor semantically equivalent to GT.

Scenario 2. Partial observability: partial initial and goal states + some actions in the plan + some intermediate states

The observability of each $\mathcal{O}_i$ in this Scenario is more reduced than in Scenario 1, as now the initial/goal/intermediate states and the actions are incomplete.

We use again the example introduced above, but now the goal in ρ_i is not totally observed, e.g., $\mathcal{G}_i$={*(on T R), (on A Y), (ontable S)*}, and similarly for $\mathcal{I}_i$, where only the state of the blocks *T*, *A* and *S* are observed. Informally speaking, in this scenario we do not know in detail the initial state of the world or which are all the desired goals in ρ_i. Also, the observed plan π_i is incomplete, e.g., we have only observed the *stack* actions of Figure 2. Similarly, the intermediate states that are observed are partial. Let us assume again that $App(\mathcal{A})$ is as depicted in Figure 3.

Evaluation proposal and challenges. The semantic-based evaluation proposed in the Scenario 1 cannot be directly used here because $\mathcal{I}_i$ and $\mathcal{G}_i$ are not fully observed and, consequently, we cannot always define a complete planning problem ρ_{App}. A preliminary evaluation is to use the partially observed $\mathcal{I}_i$ and $\mathcal{G}_i$, and try to find a plan π_{App}. Two scenarios are possible for this evaluation:

- Scenario 2.1. If the information observed is too limited and insufficient to get a plan, no π_{App} will be found. This is the result in the current example, because nothing about the initial state of blocks *R* and *Y* are known. In absence of π_{App}, no comparative evaluation between the observed π_i and π_{App} is possible and we need to use another type of evaluation, as we will propose in Scenario 3.
- Scenario 2.2. If the information observed is sufficient to get a plan, π_{App} will be found. For example, let us consider that more predicates are initially observed, such as $\mathcal{I}_i$={*(ontable T), (ontable R), (ontable A), (ontable Y), (ontable S), (clear T), (clear R), (clear A), (clear Y), (clear S), (handempty)*}, while $\mathcal{G}_i$ remains as above. A

```
1: (pick-up T)
1: (pick-up A)
2: (stack T R)
2: (stack A Y)
```

Figure 5: A potential plan π_{App} to solve ρ_{App} in Scenario 2.2.

potential plan π_{App} that solves this new problem is depicted in Figure 5. As can be seen, both the observed plan π_i and the π_{App} reach the goals in $\mathcal{G}_i$, starting from the same $\mathcal{I}_i$, so it seems to be a fairer way to evaluate the model than a syntax-based one. This scenario shares the same important aspects regarding the semantic evaluation as in Scenario 1. Regarding the plans, it is important to note that π_{App} and π_i are different. Although the two *stack* actions in π_{App} are present in the partially observed plan π_i, they are not planned in the same order; these actions are parallel in π_{App} and sequential in π_i. This is due to the partially observed goals in $\mathcal{G}_i$ that do not induce a particular ordering. Although this entails an additional challenge in the comparison, π_{App} could be sequentialized again in exactly the same order as π_i.

Scenario 3. Zero observability: empty observation

This is the task that uses the least complete set of observations; the initial and goal states are fully unknown and there is no knowledge on the plan actions or intermediate states ($\mathcal{O}_i$ is empty). It is important to highlight that the initial and goal states were known to build the plan, but we have no observations on them or the plan. Assuming the only information that we have is $App(\mathcal{A})$, as depicted in Figure 3, no ρ_{App} can be defined from the information in $\mathcal{O}_i$ and, consequently, no plan π_{App} can be generated (similarly to Scenario 2.1). Actually, in the hypothetical case where we might come up with an artificial or synthetic plan π_{App}, it could not be evaluated without an observation $\mathcal{O}_i$ to satisfy. Therefore, the semantic evaluation used in Scenarios 1 and 2.2. cannot be applied here.

Evaluation proposal and challenges. In absence of $\mathcal{O}_i$, the only option to verify some evaluation metrics on $App(\mathcal{A})$ is to come up with some synthetic observations in $\mathcal{O}_{ev}$. Thus, we propose to create a set of $\mathcal{O}_i \in \mathcal{O}_{ev}$ with its corresponding $\mathcal{I}_i$ and $\mathcal{G}_i$, to define a new planning problem $\rho_{App} = \langle App(\mathcal{A}), \mathcal{I}_i, \mathcal{G}_i \rangle$ and find the corresponding plan π_{App}.

Each new problem ρ_{App} is characterized by the purpose of testing the model learned. From our suspicions, it is important to determine if all actions in the domain are reachable, which combination of predicates can be achieved as goals and which ones cannot, etc. If we think that $\mathcal{I}_i$ and $\mathcal{G}_i$ are correct in ρ_{App}, but no plan π_{App} can be found, is this an indication that $App(\mathcal{A})$ does not meet the tested guarantee or, on the contrary, our beliefs are false and the initial/goal state are incorrect? Unfortunately, we do not have a clear answer to this yet. In our opinion, how to define these correct planning problems is an open challenge and we are still in the process of understanding the causes of the setbacks in this scenario. Having a clear understanding of the evaluation in this scenario will allow us to apply it to Scenario 1 and 2 as well. After all, we can always discard the original $\mathcal{O}_{ev}$ and replace it by a synthetic $\mathcal{O}'_{ev}$ to broaden the scope of the evaluation.

Scenario 4. Uncertain observability

In the previous three scenarios, we have assumed that all the observations are noiseless. But in the real-world, sensors are not always available nor are able to capture flawless information: some information is observed when it should not (noise), whereas other information is not observed when it should (missing data). Therefore, observations are typically uncertain. The semantic evaluation discussed in the previous scenarios is still valid in presence of uncertain observations, but it needs to be slightly relaxed. The relaxation comes in terms of not requiring the full satisfiability of the entire $\mathcal{O}_{ev}$ because the learned model $App(\mathcal{A})$ can still be equivalent to GT even if it does not satisfy one (or more) uncertain $\mathcal{O}_i$.

Evaluation proposal and challenges. Let us consider the example above for a given $\mathcal{O}_i$. As usual, in $\mathcal{I}_i$ all the blocks are initially clear and on the table, but now $\mathcal{G}_i$=$\{$*(clear T), (holding T), (on R A), (on A Y), (on Y S), (ontable S)*$\}$. The observed plan π_i is the one depicted in Figure 2. Clearly, $\mathcal{O}_i$ is unsatisfiable because it contains contradictory information. On the one hand, we know that it is impossible to reach both *(clear T)* and *(holding T)*, i.e., these two goals are unsolvable. On the other hand, the action *(stack T R)* in π_i asserts *(clear T)* but retracts *(holding T)*, so $\mathcal{G}_i$ is not achieved. However, in absence of GT we cannot find out where the problem is. For example, is *(clear T)* a noisy observation, or is *(holding T)*, or perhaps both? Is the action *(stack T R)* a noisy observation (if we really want to achieve *(holding T)*) or are there missing actions? Unfortunately, there is not a conclusive answer to this because all options are possible when dealing with uncertainty.

If we assume that $App(\mathcal{A})$ is given by Figure 3, we can try to find a plan π_{App} for ρ_{App} to be compared with π_i. Similarly to Scenario 2, two scenarios are possible for this comparison:

- Scenario 4.1. The information observed in $\mathcal{I}_i$ and $\mathcal{G}_i$ makes ρ_{App} unsolvable, so no π_{App} can be found. In absence of π_{App}, no evaluation is possible and we need to use the type of evaluation proposed in Scenario 3.
- Scenario 4.2. The model learned in $App(\mathcal{A})$ supports the uncertain observations in $\mathcal{I}_i$ and $\mathcal{G}_i$, and a plan π_{App} is found. This is the result in the current example. A potential plan π_{App} for ρ_{App} is shown in Figure 6. All actions in π_{App} appear in π_i. Intuitively, the fact of not learning the negative effects makes it possible to achieve the entire $\mathcal{G}_i$ because *(clear T)* and *(holding T)* are simultaneously possible in a relaxed situation. As can be seen, all actions in π_{App} appear in the observed π_i, although in a different order. But, without the negative effects, π_{App} could be sequentialized in exactly the same order as π_i.

```
1: (pick-up R)
1: (pick-up A)
1: (pick-up Y)
1: (pick-up T)
2: (stack R A)
2: (stack A Y)
2: (stack Y S)
```

Figure 6: A potential plan π_{App} to solve ρ_{App} in Scenario 4.2.

Conclusions

Classical approaches on Domain Repair and Domain Learning evaluate the quality of the learned model $App(\mathcal{A})$ by comparing it with the GT. But loosely speaking, if we have access to GT (as a reference model) to compare to, there would be no need for learning or repairing. However, in real-world situations, we typically only have access to some input data in the form of observations. Therefore, in this paper we offer alternative approaches to evaluate $App(\mathcal{A})$ based on the available observations, which might be uncertain.

More specifically, this paper discusses different ways to evaluate learned models without the limitations of syntactic-based evaluations. We name these new ways as semantic-based evaluations, which are mainly oriented to assess whether the learned model captures the semantics of GT in terms of achieving the same goals, executing the same actions (perhaps in the same order) or traversing the same or similar intermediate states. Further, we explore alternatives for semantically evaluating models in the absence of the GT, where the evaluation is limited to the input data and the observability directly impacts the evaluation. We think these alternatives raise new problems and challenges, unsolved yet, in the evaluation of learned planning models.

From the exploratory study of scenarios that we have presented, we can draw two main conclusions:

1. In the presence of observations, if the information is sufficient to generate a plan using the learned model, such a plan can be evaluated to determine if it fully satisfies the observations.
2. In the absence of observations, an alternative is to create synthetic observations to test the learned model. However, creating these planning problems correctly remains an open challenge.

Acknowledgements

This work has been partially supported by grants PID2021-127647NB-C21 and PID2021-127647NB-C22 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe" and Next Generation EU/ PRTR. Also by the Madrid Government under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M17) in the context of the V PRICIT (Regional Programme of Research and Technological Innovation). This project has been supported by the Doctoral School of UC3M.

References

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

Fikes, R.; and Nilsson, N. 1971. STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2: 189–208.

Garrido, A. 2022. Learning temporal action models from multiple plans: a constraint satisfaction approach. *Engineering Applications of Artificial Intelligence*, 108: 104590.

Garrido, A. 2023. Learning cost action planning models with perfect precision via constraint propagation. *Information Sciences*, 628: 148–176.

Gragera, A.; Fuentetaja, R.; Olaya, Á. G.; and Fernández, F. 2023. A Planning Approach to Repair Domains with Incomplete Action Effects. In *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, July 8-13, 2023, Prague, Czech Republic*, 153–161. AAAI Press.

Kambhampati, S. 2007. Model-lite Planning for the Web Age Masses: The Challenges of Planning with Incomplete and Evolving Domain Models. In *Proceedings of AAAI 2007, July 22-26, 2007, Vancouver, British Columbia, Canada*, 1601–1605. AAAI Press.

Kucera, J.; and Barták, R. 2018. LOUGA: Learning Planning Operators Using Genetic Algorithms. In *Pacific Rim Knowledge Acquisition Workshop, PKAW-18*, 124–138.

Lin, S.; Grastien, A.; and Bercher, P. 2023. Towards Automated Modeling Assistance: An Efficient Approach for Repairing Flawed Planning Domains. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, 12022–12031. AAAI Press.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL–the planning domain definition language–version 1.2. Technical report, Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Minaee, S.; Kalchbrenner, N.; Cambria, E.; Nikzad, N.; Chenaghlu, M.; and Gao, J. 2020. Deep Learning Based Text Classification: a Comprehensive Review.

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.