Planning while Learning with Anytime Sound and Complete Models

Pablo Copete[1], **Diego Aineto**[1], **Eva Onaindia**[1], **Enrico Scala**[2]

[1]Universitat Politècnica de València
[2]Università degli Studi di Brescia
{pcopgar,dieaigar,onaindia}@vrain.upv.es, enrico.scala@unibs.it

Abstract

AI planning typically assumes access to an accurate action model, which must often be learned from data. Most approaches separate learning and planning, requiring the model to converge before it can be used. In complex environments, however, convergence may be infeasible, requiring agents to act under epistemic uncertainty. We propose a framework for planning while learning that exploits sound and complete approximations of the unknown model. These approximations enable safe exploitation of known behavior and principled exploration of uncertain transitions. We outline two modes of integration between learning and planning and implement one of them. Experiments show that our approach can greatly reduce simulator interaction, enabling effective task solving well before model convergence.

Introduction

Black-box planning settings arise when an agent must solve a task without access to the underlying action model, but can interact with a simulator. In such settings, the model may be only partially identifiable, or too costly to learn before useful action is required. The challenge is therefore not simply to learn the model, but to pursue the task under epistemic uncertainty about it.

Most work on action model learning for AI Planning treats this uncertainty as a temporary obstacle: the goal is to refine the model until the planning task can be tackled reliably. Even when model learning and planning are interleaved, interaction is typically driven primarily by model improvement. In contrast, we study how to address the planning task from the outset, using partial model knowledge to guide planning while continuing to learn through interaction. Concretely, we argue for a shift from *learning models for planning* to *planning while learning models*, using principled approximations that provide useful guidance and guarantees throughout the learning process.

To support this, we build on the *Anytime Sound and Complete Action Learning* (ASCAL) framework (Aineto and Scala 2024), which maintains two complementary approximations of the true model: a *sound* model that guarantees safe execution, and a *complete* model that preserves all transitions not yet ruled out. We interpret these models as dual representations of uncertainty: the sound model enables reliable exploitation, while the complete model enables exploration of behaviors that remain plausible.

Based on this view, we propose a framework for planning while learning in which the agent exploits sound and complete model approximations to guide decision making during learning. We define two instantiations of this framework: (i) a *model-based planning* mode, where planning is performed over the learned models and validated against the simulator, and (ii) a *model-guided interaction* mode, where search is performed directly over the simulator and guided by the learned models. In this paper, we implement the former and provide a preliminary evaluation across multiple domains, measuring coverage, runtime, and interaction efficiency; the latter is left for future work. Our results indicate that leveraging both sound and complete approximations can substantially improve interaction efficiency, and can translate into earlier task solving as search difficulty increases.

Problem Setting

We consider a black-box planning setting where the true action model is unknown, but a simulator is available for querying. We focus on deterministic propositional action models with conjunctive preconditions specified in PDDL (McDermott et al. 1998).

Let F be a set of fluents, A a set of action symbols, and S the set of states induced by F. The true action model M^* is defined over F and A, but its preconditions and effects are unknown to the agent. The agent is given an initial state $s_0 \in S$, a goal condition G, the sets F and A, and access to a simulator defined over the same representation.

Interaction with the simulator is organized through *execution instructions* of the form (s, π), where s is a previously validated state and π is a finite sequence of actions. Executing (s, π) means querying the simulator for the result of applying the actions in π from state s, until either the sequence is exhausted or some action fails. The outcome of executing an instruction is a set of *demonstrations* corresponding to the executed prefix. These demonstrations can be: (i) *positive*, of the form (s, a, s'), indicating that executing action a in state s results in state s', or (ii) *negative*, of the form $(s, a, \bot)$, indicating that action a is not executable in state s.

The objective is to compute a goal-achieving action sequence through simulator interaction, without access to the

true action model. The agent must therefore plan under epistemic uncertainty using only the gathered data.

Background: ASCAL

We build on the Anytime Sound and Complete Action Learning (ASCAL) framework (Aineto and Scala 2024), which provides sound and complete approximations of an unknown action model from demonstrations.

Sound and complete models. Let $T_M \subseteq S \times A \times S$ denote the transition system induced by an action model M, i.e., the set of transitions accepted by M. At any point during learning, ASCAL maintains two models:

- A **sound model** M_S, which under-approximates the true model: $T_{M_S} \subseteq T_{M^*}$, containing only transitions that are guaranteed to be valid.
- A **complete model** M_C, which over-approximates the true model: $T_{M_C} \supseteq T_{M^*}$, containing all transitions that have not yet been ruled out by the data.

Anytime guarantees. As new demonstrations are collected, the sound and complete approximations are progressively refined, reducing uncertainty about the true model. Crucially, these approximations are *anytime*: after any number of demonstrations, the agent can derive both M_S and M_C. Thus, ASCAL does not commit to a single current hypothesis about the hidden model, but maintains lower and upper bounds on the behaviors still supported by the data.

Planning properties. The sound and complete models immediately yield the following properties.

Proposition 1 (Safe execution)**.** *Let π be a solution plan for $\langle M_S, s, G\rangle$. Then π is a solution plan for $\langle M^*, s, G\rangle$*

Proposition 2 (Optimistic completeness)**.** *If there exists a solution plan for $\langle M^*, s, G\rangle$, then there exists a solution plan for $\langle M_C, s, G\rangle$.*

Proposition 3 (Admissible guidance)**.** *Let h_{M_C} be a heuristic computed from M_C that is admissible with respect to M_C. Then h_{M_C} is admissible with respect to M^*.*

All three properties follow directly from the approximation relations $T_{M_S} \subseteq T_{M^*} \subseteq T_{M_C}$. Soundness ensures that every transition admitted by M_S is valid in the true model, so any plan found in M_S is executable in M^*. Completeness ensures that every true transition is preserved in M_C, so any true solution remains feasible in M_C. Since M_C is an over-approximation of M^*, distances to the goal in M_C are lower than or equal to those in M^*, and any admissible heuristic for M_C therefore remains admissible for the true model.

A Framework for Planning while Learning

We propose a general framework for solving planning tasks under unknown action models. The framework consists of three interleaved components: a learning module (ASCAL), a planner module, and the simulator.

Algorithm 1: ASCAL-Guided Planning Framework

Require: (F, A, s_0, G)
1: Initialize $\text{ASCAL}(F, A)$
2: Initialize $\text{PLANNER}(s_0, G)$
3: **while** true **do**
4: $\quad (M_S, M_C) \leftarrow \text{ASCAL.MODELS}()$
5: $\quad I \leftarrow \text{PLANNER.STEP}(M_S, M_C)$
6: $\quad$ **if** $I = \emptyset$ **then**
7: $\quad\quad$ **break**
8: $\quad E \leftarrow \emptyset$
9: $\quad$ **for** each instruction $i \in I$ **do**
10: $\quad\quad E \leftarrow E \cup \text{EXECUTE}(i)$
11: $\quad \text{ASCAL.UPDATE}(E)$
12: $\quad \text{PLANNER.FEEDBACK}(E)$

ASCAL. We treat ASCAL as a stateful learning module that incrementally updates its internal representation from demonstrations. At any point during learning, the module provides two models: (i) a sound model M_S, and (ii) a complete model M_C. The module exposes two operations:

- MODELS(): returns the current pair (M_S, M_C),
- UPDATE(E): incorporates a set of demonstrations E.

Planner. The planner is a stateful component initialized with the planning instance (s_0, G). It determines which execution instructions should be issued next, and maintains any internal search state required by the chosen instantiation. The planner exposes two operations:

- STEP(M_S, M_C): returns a finite set I of instructions,
- FEEDBACK(E): updates the planner's internal state based on the demonstrations collected during execution.

The planner publishes incumbent solutions as they are found, yielding an anytime framework in which learning and search can continue to refine the current incumbent.

General framework The general framework is shown in Algorithm 1. At each iteration, the agent queries ASCAL for the current models and invokes the planner to obtain a set of execution instructions. The demonstrations returned by the simulator after executing these instructions are aggregated and then used to update both ASCAL and the planner. The loop terminates when the planner returns an empty instruction set, signaling either that a final solution has already been certified or that no further progress is possible.

We next define two endpoint instantiations of the framework. In *model-based planning*, search is performed over the learned models, which serve as surrogate transition systems, while the simulator acts as a validation oracle. In *model-guided interaction*, search is performed directly over the simulator, treated as a black-box transition system, while the learned models guide node selection and expansion.

Model-based planning

Algorithm 2 gives the corresponding STEP and FEEDBACK procedures, where PLAN denotes a call to an internal planner. The planner first attempts to compute a plan under the

Algorithm 2: PLANNER (Model-based planning)

Require: STEP(M_S, M_C) and FEEDBACK(E) share internal state: SOLVED and candidate plan π_c

function STEP(M_S, M_C)
  if SOLVED **then**
    return $\emptyset$
  $\pi \leftarrow$ PLAN(M_S, s_0, G)
  if $\pi \neq \emptyset$ **then**
    PUBLISHPLAN(π)
  $\pi \leftarrow$ PLAN(M_C, s_0, G)
  if $\pi \neq \emptyset$ **then**
    $\pi_c \leftarrow \pi$
    return $\{(s_0, \pi)\}$
  return $\emptyset$
function FEEDBACK(E)
  if π_c is fully validated by E **then**
    PUBLISHPLAN(π_c)
    SOLVED $\leftarrow$ **true**

Algorithm 3: PLANNER (Model-guided interaction)

Require: STEP(M_S, M_C) and FEEDBACK(E) share internal state: FRONTIER, g, and goal G

function STEP(M_S, M_C)
  if FRONTIER $\neq \emptyset$ **then**
    $s \leftarrow \operatorname{argmin}_{s \in \text{FRONTIER}} (g(s) + h_{M_C}(s))$
    FRONTIER $\leftarrow$ FRONTIER $\setminus \{s\}$
    if $s \models G$ **then**
      PUBLISHPLAN(π_s)
      return $\emptyset$
    $\pi \leftarrow$ PLAN(M_S, s, G)
    if $\pi \neq \emptyset$ **then**
      PUBLISHPLAN$(\pi_s \cdot \pi)$
    return $\{(s, \langle a \rangle) \mid a \in \text{APPLICABLE}(M_C, s)\}$
  return $\emptyset$
function FEEDBACK(E)
  for all $(s, a, s') \in E$ **do**
    if $s' \neq \bot$ **and** $g(s) + \text{COST}(a) < g(s')$ **then**
      $g(s') \leftarrow g(s) + \text{COST}(a)$
      FRONTIER $\leftarrow$ FRONTIER $\cup \{s'\}$

sound model M_S. If a plan is found, it is published immediately as a valid incumbent. The planner then attempts to solve the task in the complete model M_C; if a plan is found, it is stored as π_c and returned as an instruction to be validated in the simulator. If no plan exists in M_C, the task is unsolvable so the planner returns the empty set. FEEDBACK publishes the candidate plan π_c if it has been fully validated by the simulator and sets a SOLVED flag, so that the next call to STEP returns the empty set, signaling termination. For simplicity, Alg. 2 replans from the initial state, although replanning from the latest validated state is also possible.

This mode relies directly on the sound and complete guarantees above. By Prop. 1, plans found in M_S can be published immediately as valid incumbents. By Prop. 2, planning in M_C preserves all true solutions, so candidate plans from M_C can drive exploration and learning through simulator validation. Thus, M_S supports safe exploitation, while M_C supports optimistic exploration. When the internal planner is optimal, published solutions improve monotonically; any fully validated optimal plan for M_C is also optimal for the true model and the planner can terminate.

Model-guided interaction

Algorithm 3 gives the corresponding STEP and FEEDBACK procedures. This instantiation can be viewed as an A*-like search over simulator-validated states, guided by a heuristic h_{M_C} derived from the current complete model. Since learning refines M_C over time, h_{M_C} is dynamic rather than fixed. The planner maintains a frontier of validated states together with their g values. At each iteration, STEP extracts a frontier state minimizing $g(s) + h_{M_C}(s)$. If the extracted state s satisfies the goal, the planner publishes the validated plan to that state π_s and signals termination. Otherwise, it first attempts to compute a suffix plan π for $\langle M_S, s, G \rangle$; if successful, the planner publishes the concatenated plan $\pi_s \cdot \pi$ as a valid incumbent. It then expands s by returning all one-action probes allowed by the complete model, namely the instruction set $\{(s, \langle a \rangle) \mid a \in \text{APPLICABLE}(M_C, s)\}$. The simulator determines which probes succeed and which fail. Positive demonstrations (s, a, s') generate validated successor states that are inserted into FRONTIER through FEEDBACK; negative demonstrations $(s, a, \bot)$ do not generate successors, but still refine ASCAL's models, possibly changing h_{M_C} values. Dead-end information is handled implicitly through the heuristic: $h_{M_C}(s) = \infty$ signals that the state is unsolvable in M_C, and ensures that it will not be selected while any frontier state with finite evaluation remains.

This instantiation is grounded in the properties of the complete model. Since $T_{M^*} \subseteq T_{M_C}$, probing all actions applicable in M_C preserves completeness while avoiding actions already ruled out by the data, and any state unsolvable in M_C can be treated as a dead-end. The same over-approximation also makes M_C a relaxation of the true model, so it can be used to derive heuristic guidance. By Prop. 3, if h_{M_C} is admissible for M_C, then h_{M_C} is also admissible for the true model, enabling informed search over the simulator while retaining the usual optimality guarantees. The sound model M_S supports safe exploitation from validated states: if π solves $\langle M_S, s, G \rangle$, then by Prop. 1 the concatenated plan $\pi_s \cdot \pi$ is a valid plan for the true model.

Empirical Evaluation

We report a preliminary empirical evaluation of the model-based planning mode (Alg. 1-2) using LAMA off-the-shelf as internal planner (Richter and Westphal 2010) and stopping when the first incumbent is published. Our objective is to assess whether our framework improves task solving relative to uninformed interaction. To isolate this question, we compare ourselves against a blind-search baseline implemented as uniform-cost search over the simulator.

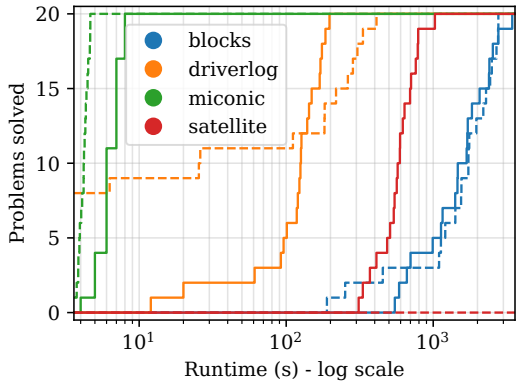

Figure 1: Cumulative coverage over time for ASCAL-guided planning (solid line) and blind search (dashed).

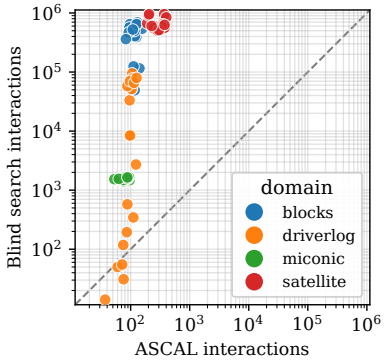

Figure 2: Simulator interactions required with ASCAL-guided planning and blind search (log scale).

Setup. We evaluated the approach on four domains: BLOCKS, DRIVERLOG, MICONIC, and SATELLITE. For each domain, we considered a suite of 20 problem instances. All instances were solved on a 12th Gen Intel(R) Core(TM) i9-12900KF @ 3.20GHz running Ubuntu 22.04 LTS, with memory and time limits of 8GB and 60 min, respectively. Code and benchmarks are available at Code and benchmarks are available at https://github.com/pablocopete/ascal-guided-planning.

Coverage over time. Figure 1 shows cumulative coverage as a function of runtime. The results are domain-dependent. In MICONIC, blind search dominates throughout, suggesting that in simple domains the overhead of learning and guidance may not pay off. In BLOCKS, both approaches are broadly competitive. In DRIVERLOG, blind search solves the easiest instances earlier, but ASCAL-guided planning overtakes it on harder problems. This indicates that learned guidance becomes more useful as difficulty increases. The strongest contrast appears in SATELLITE, where blind search solves no instance within the time limit, while ASCAL-guided planning achieves full coverage. This is likely due to the large branching factor of the domain, which makes uninformed interaction ineffective and increases the value of learned guidance.

Interaction efficiency. Figure 2 compares the simulator interactions required for each instance, measured as the number of actions executed in the simulator. With the exception of a few easy DRIVERLOG instances, all points lie above the diagonal, indicating consistent interaction-efficiency gains over blind search. In most cases, the reduction ranges from one to four orders of magnitude. Thus, blind search is competitive only when interactions are cheap and instances are easy, whereas ASCAL-guided planning uses simulator interaction much more effectively as difficulty grows. When interactions are expensive, these gains are likely to outweigh its computational overhead.

Solving before convergence. We additionally verified that, in all reported runs, tasks were solved before the learned model converged to the true action model. This shows that success was achieved under persistent model uncertainty, rather than after effectively recovering the true model. The empirical results therefore validate the intended use case of the framework: planning while learning, not planning after convergence.

Related Work

A large body of work studies action model learning from observed executions, aiming to infer action preconditions and effects from traces (Yang, Wu, and Jiang 2007; Amir and Chang 2008; Zhuo et al. 2010; Cresswell, McCluskey, and West 2013; Mourão et al. 2012; Zhuo and Kambhampati 2013; Stern and Juba 2017; Aineto, Jiménez, and Onaindia 2019; Verma, Marpally, and Srivastava 2021; Xi, Gould, and Thiébaux 2024; Bachor and Behnke 2024; Gösgens, Jansen, and Geffner 2025; Lamanna et al. 2025).

Recent approaches interleave learning and planning more tightly. CLAP alternates planning, execution, and model refinement (Karia et al. 2024), but in our setting it effectively follows a learn-then-plan pattern, with active learning preceding planning until convergence. By contrast, we plan throughout learning under persistent uncertainty. PLEX formulates a planned exploration framework for learning action models, where the agent generates its own planning problems and refines an over-general hypothesis through counterexamples (Mehta, Tadepalli, and Fern 2011). However, it remained a theoretical framework and was not developed as a practical planning system. Related ideas have also been explored in online RL through optimistic symbolic model estimates (Sreedharan and Katz 2023); unlike our approach, these methods do not maintain a complementary sound model for safe planning during learning.

Conclusions

We proposed a framework for planning under unknown action models using sound and complete approximations of the hidden model throughout learning. Our model-based planning instantiation reduces simulator interaction substantially and improves performance on harder tasks, showing that useful planning need not wait for model convergence. Future work includes fully developing the model-guided interaction mode and extending the framework to numeric planning, where ASCAL has recently been generalized.

References

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

Aineto, D.; and Scala, E. 2024. Action Model Learning with Guarantees. In *KR*, 801–811.

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

Bachor, P.; and Behnke, G. 2024. Learning planning domains from non-redundant fully-observed traces: theoretical foundations and complexity analysis. In *AAAI*, 20028–20035.

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

Gösgens, J.; Jansen, N.; and Geffner, H. 2025. Learning lifted strips models from action traces alone: A simple, general, and scalable solution. In *ICAPS*, 189–197.

Karia, R.; Verma, P.; Speranzon, A.; and Srivastava, S. 2024. Epistemic exploration for generalizable planning and learning in non-stationary settings. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 310–318.

Lamanna, L.; Serafini, L.; Saetti, A.; Gerevini, A. E.; and Traverso, P. 2025. Lifted action models learning from partial traces. *Artificial Intelligence*, 339: 104256.

McDermott, D.; et al. 1998. The planning domain definition language manual. Technical report, Technical Report 1165, Yale Computer Science, 1998.(CVC Report 98-003).

Mehta, N.; Tadepalli, P.; and Fern, A. 2011. Autonomous learning of action models for planning. *Advances in Neural Information Processing Systems*, 24.

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

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.*, 39: 127–177.

Sreedharan, S.; and Katz, M. 2023. Optimistic Exploration in Reinforcement Learning Using Symbolic Model Estimates. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems*, volume 36, 34519–34535. Curran Associates, Inc.

Stern, R.; and Juba, B. 2017. Efficient, Safe, and Probably Approximately Complete Learning of Action Models. In *IJCAI*, 4405–4411.

Verma, P.; Marpally, S. R.; and Srivastava, S. 2021. Asking the right questions: Learning interpretable action models through query answering. In *AAAI*, 12024–12033.

Xi, K.; Gould, S.; and Thiébaux, S. 2024. Neuro-symbolic learning of lifted action models from visual traces. In *ICAPS*, 653–662.

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

Zhuo, H. H.; and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *IJCAI*, 2444–2450.

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