Learning Action Models from Partially Ordered Action Traces

Chunjie Liu[1] **and Patrik Haslum**[1]

[1]Australian National University
Canberra, Australia
{firstname.lastname}@anu.edu.au

Abstract

Learning action models is a fundamental task in the field of automated planning. In many realistic settings, available data is incomplete or only partially observable, making the learning problem even harder. This paper addresses learning action models from partially ordered action traces, in which the order of actions is only partially observed. We introduce *POLOCM*, a novel method that extends the *LOCM2* algorithm with a Binary Integer Programming formulation to recover missing order information from partially ordered action traces. Experiment results show that this is effective in overcoming the additional challenge due to missing order information, while the overall model quality remains constrained by *LOCM2*'s inherent limitations.

Introduction

In Artificial Intelligence (AI), planning involves generating sequences of actions, that guide the system from an initial state to a specified goal. To create these action sequences, AI planners depend on action models – formal descriptions of actions in terms of their preconditions and effects, typically specified in declarative planning languages like the Planning Domain Definition Language (PDDL) (McDermott et al. 1998). However, manually constructing these models requires time and domain expertise.

A way to address this challenge is to automate action model acquisition, that is, to learn the action model from observations of plans or plan executions. The majority of current approaches depend on observation of both actions and corresponding (partial) state information to infer action preconditions and effects (Arora et al. 2018). In many potential applications, however, only actions, or events, can be observed. Examples include business process mining (van der Aalst 2013), temporal databases (Boselli et al. 2014), diagnosis of discrete-event systems (Cordier and Thiébaux 1994; McIlraith 1994) and narrative understanding (Lindsay et al. 2017; Li et al. 2024). Many realistic settings also feature incomplete or noisy observations, such as imperfect observation of the order of events (Haslum and Grastien 2011; Olmo, Sreedharan, and Kambhampati 2021; Li, Haslum, and Cui 2023) or missing or incorrect action arguments (Boselli et al. 2014; Li et al. 2024).

Recent advances in planning model learning have addressed settings in which state, and in a few cases also action, observations are incomplete or noisy (Aineto, Celorrio, and Onaindia 2019; Zhuo and Kambhampati 2013; Zhuo, Peng, and Kambhampati 2019; Grand, Pellier, and Fiorino 2022). However, to the best of our knowledge, only the *LOCM*/*LOCM2* (Cresswell, McCluskey, and West 2009, 2013; Cresswell and Gregory 2011) and the recently proposed SIFT (Gösgens, Jansen, and Geffner 2024) algorithms are able to learn planning models from observation of action traces only, without requiring any state information, and these algorithms all depend on observing the complete and total order of the action sequence. This can be a significant limitation to their applicability: For example, in one of the two discrete-event diagnosis benchmark domains presented by Haslum and Grastien (2011), the average flex value[1] of the partially ordered observations is 0.23. Li et al. (2023) after extracting events and their order from narrative text, report "on average, [...] 2.7 subnetworks per document, with an average size of 22 events", implying average flex values in excess of 0.5.

To bridge this gap, we present *POLOCM*, which extends the *LOCM*/*LOCM2* algorithms to learn action models from action traces where the order of actions is only partially observed. We mean by this that although the executed plan is a sequence, we may not have observed the exact order of all actions in the sequence. It is not the same as assuming that actions whose order is not observed occur in parallel, or that any ordering of them results in a valid plan. In principle, *POLOCM* can be applied even if the input is a set of completely unordered action instances, but in this case learned model will be one, arbitrary, out of many consistent models. In practice, we show that *POLOCM* can learn a model that is nearly as accurate as what would have been learned from observing the completely ordered action traces for moderate flex values (up to around 0.6). In this paper, we assume that all actions in the plan, and their arguments, are known; only the information about their order is partial. We also assume that any ordering information that is observed is correct. Learning from partial observation of the actions themselves, including their arguments, has been addressed in prior work (Gregory, Lindsay, and Porteous 2017), and

[1]Flex is a commonly used measure of the degree of ordering in a partially ordered set. It ranges between 0 and 1, where a total order has a flex value of 0 and an unordered set a flex value of 1. It is formally defined in the "Background" section later in the paper.

integrating both types of incompleteness is straightforward.

The key principle of learning from action traces alone is to find a model that admits the observed action traces, which are known to be valid, and minimises the set of unobserved action sequences admitted by the model[2]. This corresponds to the principle of maximum likelihood estimation. We extend this principle to the resolution of unobserved ordering of actions in the input traces.

LOCM and *LOCM2* identify pairs of consecutive transitions for each object type, or sort, from the ordered action traces, and use these to construct action models. This property of the algorithms allows us to concentrate on finding a minimal set of consecutive transition pairs for each sort that is sufficient to explain the partially ordered action traces, thus maximizing the fit between the resulting action model and the observations. *POLOCM* solves a Binary Integer Program (BIP) to produce a minimal transition matrix for each sort, which serve as input for *LOCM*/*LOCM2* to construct action models in PDDL format.

We evaluate the approach experimentally on multiple benchmark domains, demonstrating that in a majority of instances, *POLOCM* is able to recover an action model from partial-order action traces that is as good as the model obtained by *LOCM2* on the corresponding totally ordered action traces, even in some cases with a very high degree of missing order information.

Related Work

SIFT (Gösgens, Jansen, and Geffner 2024) is the only approach outside the LOCM family that learns action models solely from action traces. It employs a feature-based learning framework to systematically extract domain predicates and action schemas by analyzing the effects of actions on predicate patterns. Compared to *LOCM* and *LOCM2*, SIFT is able to identify predicates of any arity, and offers a form of guarantee of completeness in the limit of sufficient training examples. It relies, however, on the observed action traces being totally ordered.

LC_M (Gregory, Lindsay, and Porteous 2017) is closely related to our work, as both approaches build upon the *LOCM* algorithms and rely solely on action traces. LC_M learns action models from traces with partially observed actions that may contain noisy or missing arguments. However, it still assumes that the actions are totally ordered, which makes its problem setting different from ours. LC_M splits input plans into smaller, complete fragments and reconstructs missing parts using constraints. Noisy data is identified through transition matrices and state parameter consistency checks, followed by hypothesis testing to correct errors. While LC_M uses a constraint-based approach to infer missing information and correct errors in action sequences, our method integrates constraints to reconstruct the ordering between actions.

Unlike *POLOCM* and LC_M, which learn solely from action traces, other action models learning approaches that handle incompleteness and noise rely on some state observations: Lamanna et al. (2025) introduce an approach for learning minimal action models from partially observed state and action traces, similar in spirit to *POLOCM*, by minimizing the number of transitions required to explain the observed data. It iteratively applies a sound and complete set of completion rules to derive all minimal models which contain only the necessary effects to explain observed state changes.

AMLSI (Grand, Pellier, and Fiorino 2022) uses grammar induction and supervised learning. It requires as input both valid and invalid action sequences, which are generated via random walks and labeled based on partial and noisy state observations, or marked as "failure" when execution is not possible. While AMLSI, like POLOCM, also employs finite state machines to model state transitions, it requires input actions to be labeled as valid or invalid. IRALe (Rodrigues, Gérard, and Rouveirol 2010) also uses randomly generated action traces but focuses on complete but noisy state observations. It employs an online active learning algorithm that iteratively explores and refines the action model.

The ALICE framework (Mourão et al. 2012) learns STRIPS action models from partially observed and noisy state data by using classifiers to predict action effects and then deriving STRIPS rules. It observes actions failing (due to unsatisfied preconditions). NOLAM (Lamanna and Serafini 2024) also addresses noisy sensor-observed state–action traces, using a probabilistic graphical model to infer action preconditions and effects.

AMAN (Zhuo and Kambhampati 2013) is another graphical model-based approach for learning action models from noisy state–action traces, assuming each action has a chance of being noisy. It uses a policy-gradient method to predict correct traces, updates the domain model based on a reward function, and iteratively refines the predictions and model. *AMDN* (Zhuo, Peng, and Kambhampati 2019) extends it by also addressing disordered traces. It constructs constraints related to action disorder, parallelism, and noise, which are solved using a MAX-SAT framework to generate action models.

Le, Juba, and Stern (2024) propose two algorithms PI-SAM and EPI-SAM for learning action models from partially observed trajectories, in which actions are fully observed but states are partial. Both algorithms learn safe action models, meaning any plan generated from the learned model is guaranteed to execute successfully and achieve its intended goals.

Background

STRIPS

STRIPS (STanford Research Institute Problem Solver) is a widely-used action representation formalism in classical planning (Fikes and Nilsson 1971). We follow it to define actions as tuples $\langle a, pre, add, del \rangle$, where:

- **Action name** a: The name of the action with zero to more parameter.

[2] *LOCM* and *LOCM2* both aim to construct a model that minimises admitted unobserved action sequences, though neither offers any guarantee that the model is minimal in this respect. The SIFT algorithm, we believe, achieves minimality in the limit of increasing observations. A variant *LOCM*-like algorithm achieves minimality for a restricted class of domains.

- **Preconditions** $pre(a)$: A set of predicates that must hold true for the action a to be applicable.
- **Add effects** $add(a)$: The set of predicates that become true after the action is applied.
- **Delete effects** $del(a)$: The set of predicates that become false after the action is applied.

Given a state st, an action a can be applied if all preconditions $pre(a)$ are satisfied in st. The resulting state st' is derived by updating st according to the effects of a: adding the predicates in $add(a)$ and removing those in $del(a)$.

The *LOCM2* Algorithm

The *LOCM2* algorithm learns action models from observation of only action traces, without requiring any state observations or background knowledge. The following summary of the algorithm is necessarily very brief; for a complete description, we refer to the original presentations (Cresswell, McCluskey, and West 2013; Cresswell and Gregory 2011).

Given an action trace of length N,

$$[a_i(o_{i,1},...o_{i,r[a_i]}),\ i \in [1,N]]$$

a_i is the ith action's name, $r[a_i]$ its arity, and $o_{i,1},...o_{i,r[a_i]}$ its argument objects. We say that the ith action contains an object o ($o \in O_{a_i}$) if $o \in \{o_{i,1},...o_{i,r[i]}\}$. *LOCM2* assumes the set of objects is partitioned into disjoint subsets, called **sorts**. Objects of the same sort (i.e., type) have the same behavior, meaning they undergo identical state transitions in response to the same actions, we use $o \in O_s$ to denote object o belongs to sort s. Action arities and parameter types are assumed to be consistent across traces, that is, if $a_i = a_j$, then $r[a_i] = r[a_j]$, and for each $l \in [1, r[a_i]]$, objects $o_{i,l}$ and $o_{j,l}$ belong to the same sort. The sorts of objects in a given action trace can be inferred from this: two objects that appear in the same argument position of different instances of an action have the same sort. We use O_U and S_U to denote the universe of objects and sorts, respectively.

We will call $a.k$ an *object event* (or event), for an action a and $k \in [1, r[a]]$. It represents the event that occurs to the k^{th} object in the execution of a.

Two actions a_i and a_j in an action trace are *consecutive with respect to object* $o \in O_s$ iff:

- $i < j$
- Both actions have o as a common parameter, that is, $o = o_{i,k} = o_{j,l}$ for some $k \in [1, r[a_i]], l \in [1, r[a_j]]$
- There is no action a_p such that $i < p < j$ and $o \in O_{a_p}$.

The corresponding events $a_i.k$ and $a_j.l$ are called consecutive events. For each sort s, *LOCM2* collects all consecutive events of the sort in a **transition matrix** T^s, where $T^s_{ik,jl} = 1$ if $a_i.k$ and $a_j.l$ are consecutive. It then constructs one or more finite state machines (FSMs) for each sort, representing the state and transitions of objects of the sort: For any two events $a_i.k$ and $a_j.l$ that are consecutive, the end state of $a_i.k$ is unified with the start state of $a_j.l$, i.e. $\text{end}(a_i.k) = \text{start}(a_j.l)$. The FSM states are then parameterized by checking if all consecutive transitions through a state share another common argument. This process may overgeneralize, i.e., the created FSM may accept event sequences that are not present in the observed action traces. To address this, *LOCM2* searches for subsets of the events of a sort such that an FSM constructed over the subset covers "holes" in the matrix – gaps that would otherwise lead the FSM to overgeneralize. Each FSM created from a subset of events must be validated against all observed event sequences for the sort. The resulting set of FSMs represent how objects of the sort behave over time, where states capture preconditions and effects, and transitions represent actions that move objects between these states.

A STRIPS domain is created from the set of FSMs by representing each state of each FSM as a predicate, with a parameter for the object sort and additional parameters for any parameters of the state. Preconditions of action a are $\text{start}_m(a.k)(x_k, \ldots)$, where x_k is the kth parameter of a, for each FSM m of the sort of x_k that the event causes a transition in. The effects $\neg\,\text{start}_m(a.k)(x_k, \ldots)$ and $\text{end}_m(a.k)(x_k, \ldots)$ are added to a if $\text{start}_m(a.k) \neq \text{end}_m(a.k)$.

Note that although *LOCM2* was presented as learning from action traces, it does in fact not require the sequence of actions to be completely known: it requires only the sort transition matrix, for each sort, and the pairs of actions that are consecutive with respect to some objects. We make use of this property of the algorithm, by reconstructing from a partially ordered action trace only the required inputs, without needing the reconstruct a total ordering of all actions.

Partial-Order Action Trace

A partial-order action trace (*POAT*) is a tuple $\langle \mathcal{A}, \mathcal{C} \rangle$, where $\mathcal{A}$ is the set of action instances in the trace and $\mathcal{C}$ is the set of ordering constraints between action instances in $\mathcal{A}$. We assume that each action instance in $\mathcal{A}$ is uniquely identifiable (e.g., by the distinct superscript in Example 1). For actions $a_1, a_2 \in \mathcal{A}$, an ordering constraint, written $a_1 \prec a_2 \in \mathcal{C}$, indicates that action a_1 happens before action a_2. If $a_1 \prec a_2$ or $a_2 \prec a_1$, we say that a_1 and a_2 are comparable. The set of ordering constraints $\mathcal{C}$ can also be represented as an **adjacency matrix** I, where

$$I_{ij} = \begin{cases} 1 & \text{if } a_i \prec a_j \\ 0 & \text{if } a_i \not\prec a_j \end{cases} \quad \forall i,j \in [1, |\mathcal{A}|]$$

Note that $a_i \not\prec a_j$ means either $a_j \prec a_i$, or a_i and a_j are not ordered.

Definition 1 *A POAT*$\langle \mathcal{A}, \mathcal{C} \rangle$ *is* strict *if all* $a_1, a_2, a_3 \in \mathcal{A}$ *satisfy the following:*

- *Transitivity:* $(a_1 \prec a_2) \wedge (a_2 \prec a_3) \implies a_1 \prec a_3$
- *Asymmetry:* $a_1 \prec a_2 \implies a_2 \not\prec a_1$
- *Irreflexivity:* $a_1 \not\prec a_1$

These properties can be enforced on the adjacency matrix I for $i, j, x \in [1, |\mathcal{A}|],\ i \neq j,\ x \neq i, j$ as follows:

- Transitivity: $I_{ij} + I_{jx} \leq I_{ix} + 1$
- Asymmetry: $I_{ij} + I_{ji} \leq 1$
- Irreflexivity: $I_{ii} = 0$

Example 1 *Consider the following partial-order action trace:*

$$\begin{tikzcd}
do^1(o_1,o_2) \arrow[rr] \arrow[dr] & & undo^2(o_1) \arrow[dr] & & do^4(o_1,o_3) \\
& get^5(o_2) & & get^3(o_3) &
\end{tikzcd}$$

(The superscript is a unique identifier for each action instance.) We have $O_U = \{o_1, o_2, o_3\}$, and we can infer that o_2 and o_3 are of the same sort, so there are two sorts $s_1 = \{o_1\}, s_2 = \{o_2, o_3\}$. The set of ordering constraints is $\mathcal{C} = \{do^1 \prec undo^2, do^1 \prec get^5, undo^2 \prec get^3\}$, and the corresponding adjacency matrix is

I	do^1	$undo^2$	get^3	do^4	get^5
do^1	0	**1**	1	?	**1**
$undo^2$	0	0	**1**	?	?
get^3	0	0	0	?	?
do^4	?	?	?	0	?
get^5	0	?	?	?	0

(Entries in bold are given in $\mathcal{C}$; remaining values are derived from partial-order properties. ? stands for unknown values.)

A partial-order object trace (*POOT*) is a tuple $\langle \mathcal{T}^o, \mathcal{C}^o \rangle$, where $\mathcal{T}^o$ is a set of events on object o, and $\mathcal{C}^o$ is the set of ordering constraints between events in $\mathcal{T}^o$. $\mathcal{C}^o$ can also be represented as an adjacency matrix $P^o \in \{0, 1\}$, where $P^o_{ij} = 1$ means $e_i \prec e_j$, and $P^o_{ij} = 0$ means $e_i \not\prec e_j$. A strict *POOT* satisfies the same properties as a strict *POAT*.

Given a *POAT* $\langle \mathcal{A}, \mathcal{C} \rangle$ and an object o such that $o \in O_a$ for some $a \in \mathcal{A}$, $\langle \mathcal{T}^o, \mathcal{C}^o \rangle$ is uniquely determined (up to renaming). We assume $\mathcal{T}^o$ and $\mathcal{A}$ are indexed by $i \in [1, |\mathcal{T}^o|]$ and $j \in [1, |\mathcal{A}|]$, respectively, and define the mapping $\phi^o : \mathcal{T}^o \to \mathcal{A}$ such that $\phi^o(i) = j$ when $e_i = a_j.k$ and $o = o_{j,k}$.

Example 2 *The adjacency matrices of the POOTs for each of the three objects in the trace from Example 1 are:*

P^{o_1}	$do^1.1$	$undo^2.1$	$do^4.1$
$do^1.1$	0	1	?
$undo^2.1$	0	0	?
$do^4.1$	?	?	0

P^{o_2}	$do^1.2$	$get^5.1$
$do^1.2$	0	1
$get^5.1$	0	0

P^{o_3}	$get^3.1$	$do^4.2$
$get^3.1$	0	?
$do^4.2$	?	0

From the definition of consecutive events, we have:

Lemma 1 *Given the adjacency matrix I of a POAT$\langle \mathcal{A}, \mathcal{C} \rangle$, and P^o of a POOT$\langle \mathcal{T}^o, \mathcal{C}^o \rangle$, $o \in O_U$, then for any $i, j \in [1, |\mathcal{T}^o|]$: $P^o_{ij} = I_{\phi^o(i)\phi^o(j)}$*

This states that for any two actions a_i and a_j that have a shared object o, if a_i precedes a_j, then for the two events $e_{i'} = a_i.k$ and $e_{j'} = a_j.l$, such that $o = o_{i,k} = o_{j,l}$, $e_{i'}$ also precedes $e_{j'}$.

Finally, we will use the *flex* measure of a partial order to quantify the degree of ordering. It is defined as follows (Muise, Beck, and McIlraith 2016):

Definition 2 *Given a POAT $\langle \mathcal{A}, \mathcal{C} \rangle$,*

$$flex(\langle \mathcal{A}, \mathcal{C} \rangle) = 1 - \frac{|\mathcal{C}^+|}{\sum_{i=1}^{|\mathcal{A}-1|}}$$

where $|\mathcal{C}^+|$ denotes the number of comparable action pairs (i.e. the sum of the I matrix), and $\sum_{i=1}^{|\mathcal{A}-1|}$ is the total number of possible action pairs over the trace.

Given this definition, a **fully unordered trace** will have a flex value of 1, while a **totally ordered trace** will have a flex value of 0. The flex of the *POAT* in Example 1 is 0.6.

POLOCM

POLOCM extends the *LOCM* and *LOCM2* algorithms to work on partial-order input, exploiting the property of the algorithm that only an ordering of events on each object in the trace is needed, not a total ordering of all actions. As shown in Example 2, even a loosely ordered action trace can impose a nearly total order of the events that affect each object, but this is of course not guaranteed. A simple approach to dealing with the remaining partial order is to assume that every possible ordering of the observed actions, i.e., every linearisation of the partially ordered input traces, is a valid action sequence. (We use this simple approach as a baseline for comparison with *POLOCM* in the experimental evaluation later in the paper.) This, however, is likely to result in a domain model that is much more permissive than required, i.e., that is likely to admit many more action sequences that are in fact not valid. Because our aim is to find a domain model that explains the observed partially ordered action traces and admits a minimal set of unobserved traces, we instead resolve the remaining order uncertainty so that the total number of consecutive event pairs (or transitions), across all sorts, is minimised. The intuition behind this choice is to maximize the fit between the domain model and the observations, under the fundamental assumption that different objects of the same sort behave the same way.

In Example 2, events $do.2$ and $get.1$ on object o_3 are unordered, so both pairings of the events could be consecutive. However, for object o_2, which is of the same sort as o_3, we have observed $do.2 \prec get.1$, so we resolve the uncertainty in the o_3 object trace in the same way. If, on the other hand, we did not have the observations of this order for o_2, the choice between $do.2 \prec get.1$ and $get.1 \prec do.2$ would be arbitrary, but would still be resolved the same way for o_3 and o_2 both.

We formulate the problem of finding a completion of each of the partial-order object traces, that are mutually consistent and consistent with the constraints observed in the input action traces, and that minimize the total set of consecutive events across all sorts, as a binary integer program (BIP). We conjecture that the problem is NP-hard (it is reminiscent of a hitting set problem).

For simplicity, we present the formulation using a single input trace, though the method generalizes naturally to multiple traces by aggregating individual object transitions from each trace into the final sort transition matrix.

BIP Variables Given a *POAT* $\langle \mathcal{A}, \mathcal{C} \rangle$, we can encode the global ordering constraints of the action trace using a matrix I of binary variables. Likewise, the *POOT* for each object $o \in O_U$ is represented in a matrix P^o. However, partial-order information alone is insufficient to determine consecutive events. Thus, we introduce the object-level transition matrix F^o, where $F^o_{ij} \in \{0,1\},\ i,j \in \mathcal{T}^o, i \neq j$. $F^o_{ij} = 1$ iff the events e_i and e_j are consecutive, that is, iff $e_i \prec e_j$ and there exists no $e_i \prec e' \prec e_j$. The constraints on F^o and P^o (described below) ensure that when fully assigned, the two order the events in $\mathcal{T}^o$ into a sequence.

Finally, the transitions across all objects of each sort are aggregated into a sort-level transition matrix T^s, where $T^s_{ij} \in \{0,1\},\ i,j \in m[s],\ s \in S_U$, $m[s]$ is the number of events for sort s. $T^s_{ij} = 1$ iff e_i and e_j are consecutive in the trace of some object $o \in O_s$. T^s thus captures event transitions independent of specific objects.

Example 3 *The transition matrices of the two sorts in Example 1 are:*

T^{s_1}	$do.1$	$undo.1$
$do.1$	?	1
$undo.1$	?	0

T^{s_2}	$do.2$	$get.1$
$do.2$	0	1
$get.1$	?	0

Note the transition $\langle do.1, do.1\rangle$ in T^{s_1} is undecided because two $do.1$ events occur on o_1, which may or may not be consecutive, given the ordering constraints of the POAT.

In the following sections, we describe the constraints that links these matrices together, to recover an ordering that is consistent with the given partial order and that induces a minimal set of necessary transitions.

Objective Function The objective of the BIP is to minimize the total number of transitions, across all sorts, required to explain the partially ordered action trace:

$$\text{minimize} \sum_{s \in S_U} \sum_{i=1}^{m[s]} \sum_{j=1}^{m[s]} T^s_{ij}$$

Strict partial-order constraints We impose the following constraints on I to ensure transitivity and asymmetry of a strict *POAT*:

$$I_{ij} + I_{ji} \leq 1 \tag{1}$$

$$I_{ij} + I_{jx} \leq I_{ix} + 1 \tag{2}$$

for all $i,j,x \in [1, |\mathcal{A}|],\ i \neq j,\ x \neq i,j$. Irreflexivity is achieved by setting $I_{ii} = 0$. We only add constraints on I to ensure it is a strict partial order, not a total order. As mentioned earlier, *LOCM2* does not require totally ordered action traces, only that we can recover the event sequences for each object in the trace, and the sort-level consecutive transitions. In Example 1, it is not necessary to sequence all five actions: for instance, it is irrelevant whether $undo^2(o_1)$ occurred before or after $get^3(o_3)$. By focusing on object-level transitions, we gain flexibility in assigning I, and reduce complexity while preserving the necessary ordering information for generating action models using *LOCM2*.

I to P Constraints The partial-order matrix P^o for each object $o \in O_U$ is constrained by the global partial order, as shown in Lemma 1:

$$P^o_{ij} = I_{\phi^o(i)\phi^o(j)} \tag{3}$$

where $i,j \in [1, |\mathcal{T}^o|], i \neq j$. This ensures the order of events on each object o is consistent with the global partial order of actions. It also forces P^o to be a strict partial order.

Full ordering constraints We enforce asymmetry on F^o:

$$F^o_{ij} \leq 1 - F^o_{ji} \tag{4}$$

This states that if event e_i directly precedes e_j, then e_j cannot directly precede e_i, while it is also possible that e_i and e_j are not adjacent, i.e., that $F^o_{ij} = F^o_{ji} = 0$.

The full order also requires that if two events are consecutive, no other event occurs between them:

$$F^o_{ij} = 1 \iff F^o_{ix} = 0 \land F^o_{xj} = 0,$$

where $\forall x \in [1, |\mathcal{T}^o|], x \neq i,j$. This is enforced in the BIP by constraining each row and column of F^o to have a sum of no more than 1:

$$\sum_{j=1}^{|\mathcal{T}^o|} F^o_{ij} \leq 1 \qquad \sum_{i=1}^{|\mathcal{T}^o|} F^o_{ij} \leq 1 \tag{5}$$

Finally, the total sum of the F^o matrix must equal $|\mathcal{T}^o| - 1$:

$$\sum_{i=1}^{|\mathcal{T}^o|} \sum_{j=1}^{|\mathcal{T}^o|} F^o_{ij} = |\mathcal{T}^o| - 1 \tag{6}$$

The constraints on row, column, and total sum ensure that all the events on o are ordered in a sequence. The total sum constraint ensures that the number of edges is exactly $|\mathcal{T}^o| - 1$, avoiding any cycles. The row and column constraints ensure that each event is connected at most once as either a predecessor or successor, thus preventing forming a tree.

P to F Constraints The connection between F^o and P^o for object o is given by

$$F^o_{ij} \leq P^o_{ij} \tag{7}$$

This follows since if $P^o_{ij} = 0$ indicating $e_i \not\prec e_j$, then e_i and e_j cannot be consecutive, so $F^o_{ij} = 0$ must hold. This ensures that F^o is consistent with P^o by permitting only valid consecutive transitions in accordance with the partial ordering constraints.

F to T Constraints Finally, we construct the transition matrix T^s of each sort s by aggregating over all F^o matrices for $o \in O_s$, based on the fact that if we observe that e_i and e_j are consecutive for any o, then they are consecutive events for s, i.e., $T^s_{ij} = 1$, if $\exists o \in O_s, F^o_{i'j'} = 1$, where $e_i = e_{i'}$ and $e_j = e_{j'}$. This is encoded with the following constraint:

$$F^o_{i'j'} \leq T^s_{ij} \tag{8}$$

Solution extraction Minimizing the total sum of transitions across all sorts ensures that a minimal set of transitions needed to explain the given trace is selected. Since the global ordering matrix I is only as constrained as necessary, the global partial order in the BIP solution may differ significantly from the ground truth. However, this does not impact the accuracy of the final transition matrix T^s for each sort s, which is our primary concern. By enforcing consistency between matrices P^o and I and between F^o and P^o, a total order of the event trace for each object is secured by the solution. T^s then aggregates over F^o for all objects $o \in O_s$,

Domain	$\vert\mathcal{S}\vert$	$\vert\mathcal{P}\vert$	m.$\mathcal{P}$	$\vert\mathcal{A}\vert$	m.$\mathcal{A}$	$\vert\mathcal{T}\vert$	$m.\mathcal{T}$
spanner	6	6	2	3	4	19	6
miconic	2	6	2	4	2	29	16
sokoban	3	4	3	2	5	18	9
childsnack	6	13	2	6	4	40	16
ferry	2	5	2	3	2	12	6
blocksworld	1	5	2	4	2	13	8
transport	7	5	2	3	5	24	7
satellite	4	8	2	5	4	41	13
floortile	4	10	2	7	4	48	15

Table 1: Statistics of the planning domains in our data set. $|\mathcal{S}|$ is the number of sorts, $|\mathcal{P}|$ the number of predicates and $|\mathcal{A}|$ the number of actions; m.$\mathcal{P}$ and m.$\mathcal{A}$ are the maximum arity of predicates and actions, respectively. $\mathcal{T}$ is the total number of transitions observed in the plan data, $m.\mathcal{T}$ is the maximum transitions by sort.

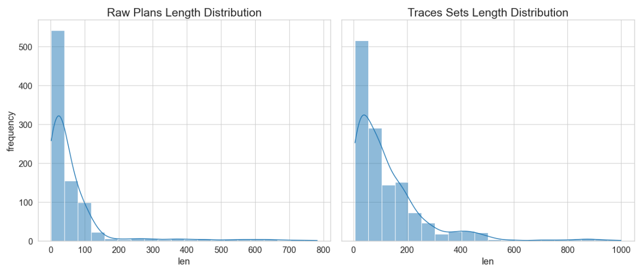


Figure 1: Distribution of the lengths of raw plans, and the total length of the learning objects (trace sets) generated by randomly sampling segments from raw plans.

providing a minimal subset of consecutive transitions that is consistent with the given *POAT*.

The sort transition matrices and object event sequences are extracted from the solution to the BIP and input to *LOCM2*, which produces a STRIPS domain model.

Example 4 *Minimizing T^{s_2} in the BIP for Example 1 will result in $\langle do.2, get.1\rangle$ being the only consecutive event pair of this sort; this implies $I_{43} = 1$, i.e., $do^4 \prec get^3$. do^4 can be ordered either before or after $undo^2$; both choices are consistent with the ordering constraints implied by minimizing T^{s_1}. Thus, there are two minimal sets of consecutive event pairs for s_1: $\{\langle do.1, do.1\rangle, \langle do.1, undo.1\rangle\}$ and $\{\langle do.1, undo.1\rangle, \langle undo.1, do.1\rangle\}$. If only one of them is actually possible, we expect that as more traces are observed, the correct pair will eventually be seen, at which point minimization will eliminate the other.*

Experiment

We start from an existing plan dataset (Chen, Trevizan, and Thiébaux 2024), covering 9 domains summarised in Table 1, with 30 plans from different instances in each domain.

To create a wider distribution of training set sizes, and increase the diversity of plan initial states, we create action trace sets of varying total length, by randomly selecting contiguous segments of varying lengths (ranging from 10 to 100) from plans in the dataset, and selecting varying numbers of these traces (small:1, medium:5, large: 10) to form a trace set for learning. The total combined length of all traces in each set is at most 1000. Figure 1 shows the distribution of plan lengths in the original dataset (left) and of total lengths over trace sets in our training collection (right).

We evaluate *POLOCM* over *POATs* of varying degrees of partial ordering. To do this, for each flex value, ranging from 0.1 to 1 and for each totally ordered action trace set, we incrementally remove the ordering constraint for comparable action pairs until the desired flex level is achieved.

We compare *POLOCM* to the *BASELINE* approach which considers all linearisations of the partial-order input traces as observations. The baseline set of consecutive events is computed without enumerating linearisations, since each pair of actions a_1 and a_2 in the *POAT* are consecutive in some linearisation unless either $a_2 \prec a_1$, or there exists an action a_3 such that, $a_1 \prec a_3 \prec a_2$. Note that the set of consecutive events computed by the baseline approach will always be a superset of that obtained from the BIP solution. For example, since both $do^4 \prec get^3$ and $get^3 \prec do^4$ occur in possible linearisations of the *POAT* in Example 1, the baseline algorithm will consider both $\langle do.2, get.1\rangle$ and $\langle get.1, do.2\rangle$ to be transitions of sort s_2. Effectively, this approach will set every undecided entry in the transition matrix of each sort to 1, resulting in a solution with a total of 5 transitions instead of the minimum of 3, shown in Example 4.

All experiments were run on a cluster node with 32 CPUs and a 48GB memory limit, with a 600-second timeout for each learning task. We use the CPLEX solver, with an individual memory limit of 4GB, and count a task as solved only if the BIP was solved to optimality.

Runtime

Figure 2 shows the percentage of problems solved within the 600-second limit, totalled over all domains (in red) and with domains divided into three groups, which show different scaling behaviour. Recall that only learning tasks for which the BIP is solved to optimality are counted as solved. The combination of medium to high disorder and increasing size of the input trace set makes solving the BIP more time consuming in all domains (as shown in the middle and right figure). However, at lower flex values (≤ 0.3) there is a clear difference between domains (in blue and green) in which difficulty does not appear to increase with the size of the input and domains (in purple) in which it does.

Reconstruction Accuracy

We measure the BIP's ability to recover ordering information from partially ordered data by the accuracy of the sort-level transition matrices extracted from the BIP solutions for partially ordered action trace sets compared to the transition matrices based on the corresponding sets of totally ordered traces. Given transition matrices extracted from the BIP solutions T^s (prediction) and from total-ordered traces $T^{s\star}$ (ground truth), accuracy is the fraction of equal entries in

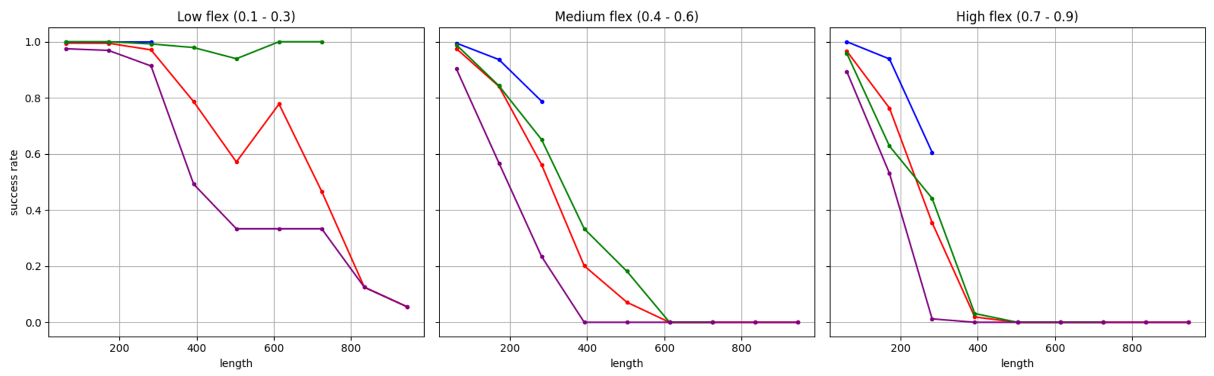
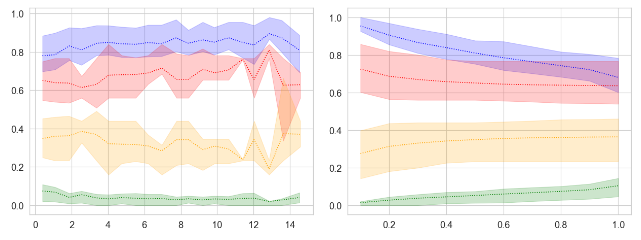


Figure 2: Percentage of problems solved within the 600s time limit by *POLOCM*, plotted against the total length of the input trace set, and across different flex levels. Blue: [spanner, minconic, sokoban, childsnack]; Green: [ferry, blocksworld, transport]; Purple: [satellite, floortile]; Red: all domains.



(a) acc/err vs. len (b) acc/err vs. flex

Figure 3: Accuracy and error rate trends with respect to (a) length of the input trace set, as a percentage of the total length of all plans for the domain; and (b) flex: *POLOCM* accuracy (blue), error rate (green); *BASELINE* accuracy (red), error rate (orange). Dotted lines represent mean values; shaded regions indicate the interquartile range (25th–75th percentile). Only learning tasks solved within the time limit are included.

T^s and $T^{s\star}$, averaged over all sorts:

$$acc = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \frac{\sum_{i,j}[T^s_{i,j} = {T^{s\star}}_{i,j}]}{|\mathcal{T}^s|^2}$$

We also measure the one-sided error rate, which is the fraction of consecutive transitions allowed in the transition matrices obtained from the BIP solution but not by the transition matrices derived from the totally ordered data:

$$err = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \frac{\sum_{i,j}[T^s_{i,j} > {T^{s\star}}_{i,j}]}{|\mathcal{T}^s|^2}$$

Figure 3 shows how accuracy and error rate vary with (a) increasing training set size, as a percentage of the total size of the plan set for the respective domain, and (b) degree of disorder (flex). *BASELINE*, which creates a more permissive model, consistency has lower accuracy, and significantly higher error rate (i.e., more false positives) than *POLOCM*. Neither approach is significantly affected by the size of the training set, but for *POLOCM* the accuracy decreases (and error rate increases) steadily with flex.

Generalization

Because the domain model is learned from observations of action traces only, the predicates it uses to define them have no relation to those of the hand-crafted domain that the training traces were generated from. This makes a direct comparison of the learned and true domain models challenging, as we cannot (automatically) map states of an instance of one domain to equivalent states of the other. Instead, we measure the *acceptance rate* of plans, both valid and invalid in the ground truth domain, by testing if, for each plan, there exists any initial state of the learned domain that it is executable from; such a state exists unless a precondition of some action in the sequence is deleted by an earlier action, which can be checked in linear time. The acceptance rate of a learned domain is simply a/n, where a is the number of plans that the domain accepts and n is the total number of test plans. We use this metric to measure the learned domain's ability to generalize by evaluating it over a larger set of traces than that from which the domain was learned. Invalid plans are created by taking a valid plan, accepted by the learned domain, and appending an action that can not be executable in the plan's end state according to the ground truth domain. The invalid action is selected at random from action schemas that appear in the accepted plan. We compare the domains learned by *POLOCM* and *BASELINE* from partial-order traces to those learned by *LOCM2* from the corresponding total-order traces.

Figure 4 shows valid and invalid acceptance rates against the training set size, as a percentage of the total size of the test plan set, and flex. As expected, more input data typically improves generalization. At moderate flex values, the acceptance rates of domains learned by *POLOCM* are close to those obtained from *LOCM2* on the corresponding totally-ordered traces, which is a limit on the best it can achieve by reconstructing the total order exactly. At higher degrees of disorder, however, domains learned by *POLOCM* become increasingly less reliable. The *BASELINE* approach, being more permissive, consistently yields high acceptance rates, for both valid and invalid plans.

Due to limited data at higher flex levels (cf. Figure 2), we

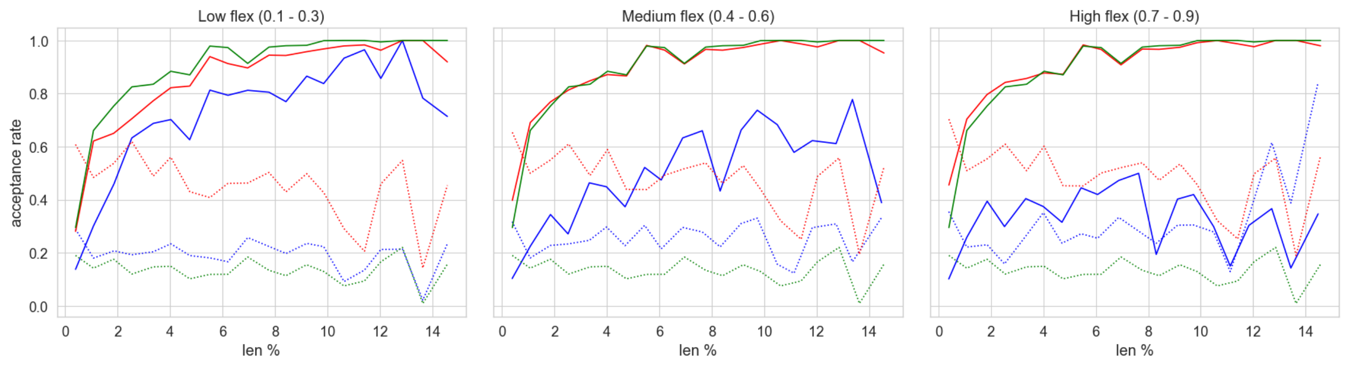


Figure 4: Mean valid (solid) and invalid (dotted) acceptance rates by trace set length as a percentage of the total test set length, across different flex levels, comparing the results of *POLOCM* (blue) and *BASELINE* (red) on partially ordered input, and *LOCM2* (green) on totally ordered input. Timed out learning tasks omitted.

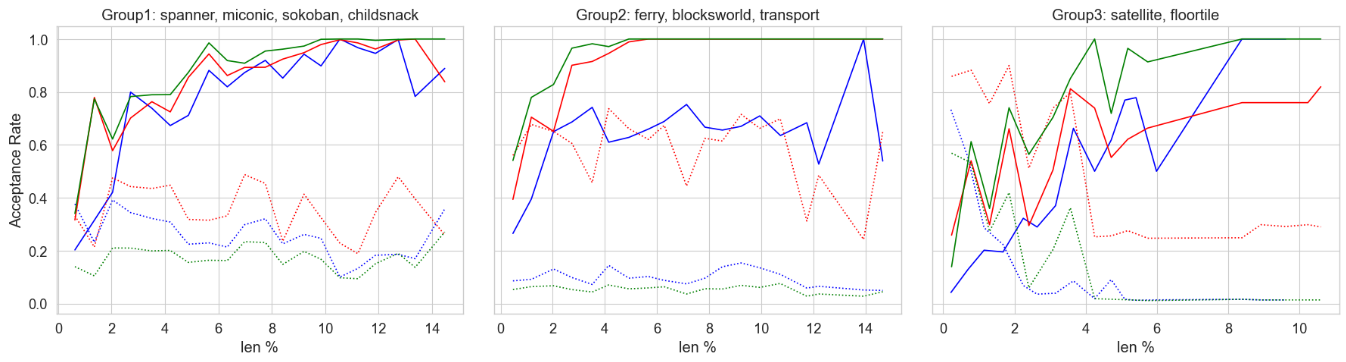


Figure 5: Mean valid (solid) and invalid (dotted) acceptance rates against input trace set length as a percentage of the total test set length under low flex levels (0.1–0.3), across the three groups of domains. From left to right: [spanner, minconic, sokoban, childsnack], [ferry, blocksworld, transport], and [satellite, floortile]. Results compare *POLOCM* (blue), *BASELINE* (red), and *LOCM2* (green). Timed out learning tasks omitted.

examine *POLOCM*'s performance in different domains at lower flex values (0.1–0.3). Figure 5 shows acceptance rates of domains divided in three groups, which exhibit different difficulty. In group 1, *POLOCM* closely matches the performance that *LOCM2* achieves with totally ordered input, and it achieves a better trade-off between valid and invalid acceptance than the more permissive *BASELINE* approach in all three groups.

Future Work

POLOCM represents a first, but significant, step toward learning planning models from partially observable action trace data, moving us closer to practical application on real-world, imperfect data sources like natural language. However, there are several further steps to explore.

First, in this paper, we have only considered learning tasks for which the BIP is solved to optimality within the time limit. Clearly, this is not a necessary limitation: The baseline approach constructs a feasible, but sub-optimal, solution to the BIP ordering problem, and of course any improvement on solution can be used in its place, even if not optimal.

Second, we have treated the given partial order on action traces as true. In some settings this is not a valid assumption, as observations of action order can be incorrect as well as missing. There are several ways to relax this assumption: Gregory et al. (2017), in LC_M, fix a threshold and discard transitions with fewer observed occurrences. In our optimisation formulation, we can treat all observations as"soft", i.e., ignorable at a cost, and optimise the two objectives of minimising model transitions and preserving the observed trace order. This raises the challenge of either defining a weighted trade-off between the two objectives, or solving a multi-objective problem.

Finally, we aim to deal with partial or noisy observations of several aspects of the input action traces, such as missing or incorrectly identified action arguments in addition to partial action order. This can be achieved by combining the different kinds of uncertainty into one optimisation problem that seeks, again, a domain model that best fits the observations while minimising spurious assumptions.

References

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artif. Intell.*, 275:

104–137.

Arora, A.; Fiorino, H.; Pellier, D.; Métivier, M.; and Pesty, S. 2018. A review of learning planning action models. *Knowl. Eng. Rev.*, 33: e20.

Boselli, R.; Cesarini, M.; Mercorio, F.; and Mezzanzanica, M. 2014. Planning meets Data Cleansing. In *Proc. 24th International Conference on Automated Planning and Scheduling (ICAPS)*.

Chen, D. Z.; Trevizan, F. W.; and Thiébaux, S. 2024. Return to Tradition: Learning Reliable Heuristics with Classical Machine Learning. In *Proc. of the 34th International Conference on Automated Planning and Scheduling (ICAPS-24)*, 68–76. AAAI Press.

Cordier, M.; and Thiébaux, S. 1994. Event-Based Diagnosis for Evolutive Systems. In *Proc. 5th International Workshop on Principles of Diagnosis (DX'94)*, 64–69.

Cresswell, S.; and Gregory, P. 2011. Generalised Domain Model Acquisition from Action Traces. In *Proc. of the 21st International Conference on Automated Planning and Scheduling (ICAPS-11)*. AAAI.

Cresswell, S.; McCluskey, T. L.; and West, M. M. 2009. Acquisition of Object-Centred Domain Models from Planning Examples. In *Proc. of the 19th International Conference on Automated Planning and Scheduling (ICAPS-09)*. AAAI.

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

Fikes, R.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artif. Intell.*, 2(3/4): 189–208.

Gösgens, J.; Jansen, N.; and Geffner, H. 2024. Learning Lifted STRIPS Models from Action Traces Alone: A Simple, General, and Scalable Solution. *arXiv*, 2411.14995v1. http://arxiv.org/abs/2411.14995v1.

Grand, M.; Pellier, D.; and Fiorino, H. 2022. An Accurate PDDL Domain Learning Algorithm from Partial and Noisy Observations. In *Proc. of the 34th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2022)*, 734–738. IEEE.

Gregory, P.; Lindsay, A.; and Porteous, J. 2017. Domain Model Acquisition with Missing Information and Noisy Data. In *KEPS 2017*. Association for the Advancement of Artificial Intelligence (AAAI).

Haslum, P.; and Grastien, A. 2011. Diagnosis as Planning: Two Case Studies. In *ICAPS'11 Scheduling and Planning Applications Workshop*.

Lamanna, L.; and Serafini, L. 2024. Action Model Learning from Noisy Traces: a Probabilistic Approach. In *Proc. of the 34th International Conference on Automated Planning and Scheduling (ICAPS-24)*, 342–350. AAAI Press.

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

Le, H. S.; Juba, B.; and Stern, R. 2024. Learning Safe Action Models with Partial Observability. In *Proc. of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)*, 20159–20167. AAAI Press.

Li, R.; Cui, L.; Lin, S.; and Haslum, P. 2024. NaRuto: Automatically Acquiring Planning Models from Narrative Texts. In *Proc. of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)*, 20194–20202. AAAI Press.

Li, R.; Haslum, P.; and Cui, L. 2023. Towards Learning Action Models From Narrative Text Through Extraction and Ordering of Structured Events. In *Proc. 36th Australasian Joint Conference on Artificial Intelligence*, volume 14472 of *LNCS*, 16–27.

Lindsay, A.; Read, J.; Ferreira, J. F.; Hayton, T.; Porteous, J.; and Gregory, P. 2017. Framer: Planning Models from Natural Language Action Descriptions. In *Proc. of the 27th International Conference on Automated Planning and Scheduling (ICAPS-17)*, 434–442. AAAI Press.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C. A.; Ram, A.; Veloso, M.; Weld, D. S.; and Wilkins, D. E. 1998. PDDL—The Planning Domain Definition Language. Technical Report DCS TR-1165, Yale Center for Computational Vision and Control.

McIlraith, S. 1994. Toward a Theory of Diagnosis, Testing and Repair.

Mourão, K.; Zettlemoyer, L.; Petrick, R. P. A.; and Steedman, M. 2012. Learning STRIPS Operators from Noisy and Incomplete Observations. In *Proc. of the 28th Conference on Uncertainty in Artificial Intelligence (UAI 2012)*, 614–623. AUAI Press.

Muise, C. J.; Beck, J. C.; and McIlraith, S. A. 2016. Optimal Partial-Order Plan Relaxation via MaxSAT. *J. Artif. Intell. Res.*, 57: 113–149.

Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2021. GPT3-to-plan: Extracting plans from text using GPT-3. *arXiv preprint arXiv:2106.07131*.

Rodrigues, C.; Gérard, P.; and Rouveirol, C. 2010. Incremental Learning of Relational Action Models in Noisy Environments. In *Inductive Logic Programming: Revised Selected Papers of the 20th International Conference (ILP 2010)*, volume 6489 of *Lecture Notes in Computer Science*, 206–213. Springer.

van der Aalst, W. M. P. 2013. Business process management: A comprehensive survey. *ISRN Software Engineering*, 2013(1): 507984.

Zhuo, H. H.; and Kambhampati, S. 2013. Action-Model Acquisition from Noisy Plan Traces. In *Proc. of the 23rd International Joint Conference on Artificial Intelligence (IJCAI-13)*, 2444–2450. IJCAI/AAAI.

Zhuo, H. H.; Peng, J.; and Kambhampati, S. 2019. Learning Action Models from Disordered and Noisy Plan Traces. *CoRR*, abs/1908.09800.