Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Learning Numeric Planning Domain Models From Positive Observations

Omar Watted[1] , **Argaman Mordoch**[1] , **Roni Stern**[1]
[1]Ben Gurion University in Be'er Sheva, Israel
{omarwa, mordocha}@post.bgu.ac.il, roni.stern@gmail.com

Abstract

Domain-independent planning algorithms require as input an action model that specifies the preconditions and effects of each action. Constructing such models is often challenging for domain experts, particularly in domains where actions involve both Boolean and numeric state variables. We address the problem of learning such hybrid action models from observations of successful action executions. A central challenge is that observing only successful executions provides no explicit evidence about which conditions are necessary for an action to be applicable. This difficulty is especially pronounced for learning numeric preconditions, for which there exist negative theoretical results concerning efficient learnability. To mitigate this limitation, we propose SAM-SVM, a novel numeric action model learning algorithm that learns numeric preconditions by heuristically simulating negative observations, i.e., possible states where actions are inapplicable. We also implemented NSAM+SVM, a hybrid algorithm that integrates SAM-SVM and NSAM, an existing conservative action model learning algorithm. Empirical evaluation demonstrates that SAM-SVM can learn more accurate action models and achieves improved planning performance compared to existing methods on standard numeric planning benchmarks, and NSAM+SVM provides the most robust behavior across most domains.

1 Introduction

A planning problem is the task of finding a sequence of actions – referred to as a *plan* – that transforms a given initial state to a state that satisfies a given goal. Domain-independent automated planners, such as Fast Downward (Helmert 2006), Fast Forward (Hoffmann 2001), and ENHSP (Scala, Haslum, and Thiébaux 2016; Scala et al. 2016; Scala et al. 2017; Li et al. 2018), are designed to solve planning problems across multiple domains. To this end, domain-independent planners require as input a domain specification that includes an *action model*, which is a formal definition of the preconditions and effects of each action. Constructing such action models is often tedious and typically requires domain experts, who may not always be available.

To address this challenge, extensive research has explored learning action models from observations (Cresswell and Gregory 2011; Cresswell, McCluskey, and West 2013; Aineto, Celorrio, and Onaindia 2019). Most prior work on action model learning has focused on domains that include only Boolean state variables. In this work, we focus on the problem of learning action models for *numeric planning*, where some state variables are numeric and may be considered by the actions' preconditions and effects.

Very few prior works exist on learning action models for numeric planning (Mordoch, Juba, and Stern 2023; Segura-Muros, Pérez, and Fernández-Olivares 2021). Existing algorithms are designed to learn solely from *positive observations*, i.e., successful action executions. This is reasonable since obtaining *negative observations* requires observing failed action executions. In real-world settings, such failures are usually rare and may even be unacceptable, as they can lead to catastrophic outcomes. For example, a robot's execution failure may cause equipment damage and incur substantial costs. Learning exclusively from positive observations can be restrictive, as many machine learning techniques rely on both positive and negative examples. Moreover, Mordoch, Juba, and Stern (2023) provided negative results on the efficiency of learning numeric preconditions from only positive observations.

To overcome these limitations, we propose SAM-SVM, an algorithm that learns numeric action models from positive observations by *simulating* negative examples. This enables the use of standard machine learning techniques that would otherwise be inapplicable. In SAM-SVM, we use an iterated version of Support Vector Machine (SVM) (Hearst et al. 1998) to create linear separators between the positive observations and the simulated negative observations. We propose two heuristics for simulating negative observations: Extended Convex Hull (ECH) and Not Used Observations (NUO). ECH extends the convex hull construction used in the Numeric Safe Action Models Learning (NSAM) algorithm (Mordoch, Juba, and Stern 2023) and samples negative points on the hyperplanes defined by the expanded hull. NUO assumes that states in which an action was not executed constitute negative examples, provided they do not overlap with the positive examples. Finally, we propose NSAM+SVM, a hybrid algorithm that uses both SAM-SVM and NSAM (Mordoch, Juba, and Stern 2023), a state-of-the-art conservative numeric action model learning algorithm.

We compared experimentally SAM-SVM with these heuristics and state-of-the-art numeric action model learning algorithms, namely NSAM (Mordoch, Juba, and Stern

2023) and PlanMiner (Segura-Muros, Pérez, and Fernández-Olivares 2021), across a set of numeric planning domains. In addition, we examined the benefits of augmenting our approach with NSAM, which provides a safety guarantee: any plan generated using the action model learned by NSAM is guaranteed to be successful under the real, unknown action model. The results demonstrate that in domains with many numeric preconditions and effects, NSAM+SVM with NUO yields the best performance in terms of problem-solving rates.

2 Background

We focus on learning planning action models from observations where the action outcomes are deterministic, the states are fully observable, and are represented as a mix of Boolean and numeric state variables. Such observations can be modeled using the PDDL2.1 language (Fox and Long 2003). A *domain* is defined by a tuple $D = \langle B, N, A, M \rangle$, where B is a finite set of Boolean variables, referred to as propositions; N is a set of numeric variables referred to as functions; A is a set of actions; and M is an action model for these actions.

An action model M is a pair of functions, pre_M and eff_M, that map actions in A to their preconditions and effects, respectively. The preconditions of an action $a \in A$, $\mathit{pre}_M(a)$, are a set of assignments over the Boolean propositions and a set of conditions over the numeric functions, specifying when the action can be applied. These conditions are of the form (ξ, Rel, k), where ξ is an arithmetic expression over N, $Rel \in \{\leq, <, =, >, \geq\}$, and k is a number. The effects of action a, denoted $\mathit{eff}_M(a)$, are a set of assignments over B and N representing how the state changes after applying a. An assignment over a Boolean proposition is either `True` or `False`. An assignment over a function $x \in N$ is a tuple of the form $\langle x, op, \xi \rangle$, where ξ is a numeric expression, and op is one of the following operations: increase ("+="), decrease ("-="), or assign (":=").

A state is the assignment of values to all variables in $B \cup N$. For a state variable $v \in B \cup N$, we denote by $s(v)$ the value assigned to v in state s. We say that an action a is applicable in a state s under the action model M, denoted $app_M(a, s)$, if s satisfies $\mathit{pre}_M(a)$. Applying a in s according to the action model M, denoted $a_M(s)$, is a state that differs from s only according to the assignments in $\mathit{eff}_M(a)$. Unless a distinction between action models is required, we omit the subscript and use $a(s)$, $\mathit{pre}(a)$, and $\mathit{eff}(a)$ instead of $a_M(s)$, $\mathit{pre}_M(a)$, and $\mathit{eff}_M(a)$.

A planning problem is defined by the tuple $\langle D, s_0, G \rangle$, where D is a domain, s_0 is the initial state, and G is the set of problem goals. G consists of assignments of values to a subset of the Boolean propositions and a set of conditions over the numeric functions. A solution to a planning problem is a *plan*, i.e., a sequence of actions $\{a_0, a_1, ..., a_n\}$ such that a_0 is applicable in s_0 and $a_n(a_{n-1}(...a_0(s_0)...))$ results in a state s_G in which G is satisfied.

Action model learning algorithms A *state transition* is a tuple $\langle s, a, s' \rangle$, where s denotes the state before the execution of a, and s' is the result of applying a in s, i.e., $s' = a(s)$. The states s and s' are commonly referred to as the *pre-state* and *post-state*, respectively. A *trajectory* is an ordered list of state transitions. Most algorithms for learning action models accept as input a set of trajectories (possibly partially observable) and output an action model. Many action model learning algorithms have been proposed over the years for *classical planning*, i.e., planning without numeric state variables (Arora et al. 2018). Some algorithms, such as LOCM (Cresswell and Gregory 2011), LOCM2 (Cresswell, McCluskey, and West 2013), and SIFT (Gösgens, Jansen, and Geffner 2025), only require the sequences of actions in the given trajectories. Other action model learning algorithms also require observing some states in the trajectories (Bolander and Gierasimczuk 2018; Mourao et al. 2012). Some algorithms reduce the action model learning problem to more generic problems for which off-the-shelf solvers exist. Examples of this type of algorithms are the Simultaneous Learning and Filtering (SLAF) algorithm (Amir and Chang 2008), which maps the action model learning problem to Boolean Satisfiability; and FAMA (Aineto, Celorrio, and Onaindia 2019), which maps it to classical planning.

Some action model learning algorithms assume full observability of the states and actions in the given trajectories. An example of such algorithms that is mostly related to our work is the Safe Action Model Learning (SAM) learning algorithm (Stern and Juba 2017; Juba, Le, and Stern 2021). SAM starts by assuming that the preconditions of an action are all the propositions and the effects are empty. Then, it uses inductive rules on the observed state transitions and determines which propositions are not preconditions for a specific action and which must be part of its effects. This conservative approach provides the SAM algorithm with the following property: every plan generated from an action model learned by SAM is guaranteed to also be a (successful) plan under the real unknown action model. An action model with this property is referred to as a *safe action model*, as it provides a sort of "execution soundness" guarantee.

The algorithms described thus far assume that all state variables are Boolean. To the best of our knowledge, only NLOCM (Gregory and Lindsay 2016), PlanMiner (Segura-Muros, Pérez, and Fernández-Olivares 2021), and NSAM (Mordoch, Juba, and Stern 2023) are capable of learning numeric aspects of action models. NLOCM extends LOCM and LOCM2 to handle numeric planning by learning action costs from trajectories using a constraint programming approach. However, it does not support learning numeric preconditions or effects. PlanMiner (PMnr) learns numeric action models from partially known and noisy plan traces. It transforms the input trajectories into datasets, one per action, where initially the features consist of the values of each proposition and numeric function in the pre- and post-states. Next, PMnr calculates the differences between each two numeric functions and applies symbolic regression to learn the expression that best represents the observed effects. The expressions representing the differences are then also added to the dataset. To create the numeric comparisons, PMnr employs a straight forward approach - for any two numeric features in the dataset, a comparison

$c \in \{<, >, =\}$ is added, and the comparisons that are consistent with the observations are added to the dataset. Finally, PMnr employs a rule-based classifier that determines the correct combination of statements (both preconditions and effects) that compose the action.

The NSAM algorithm is an action model learning algorithm from the SAM framework (Stern and Juba 2017; Juba, Le, and Stern 2021; Juba and Stern 2022; Le, Juba, and Stern 2024; Mordoch et al. 2024) that can learn numeric planning action models that are safe. NSAM assumes that the learned domains include preconditions expressed as linear inequalities and the effects are linear assignments over the previous state variables. The algorithm also assumes that the input trajectories are fully observable and noiseless. NSAM learns its action models by dividing the learning task into two parts, Boolean and numeric. To learn the Boolean preconditions and effects, NSAM employs the SAM algorithm mentioned above (Stern and Juba 2017; Juba, Le, and Stern 2021). To learn the numeric preconditions of a given action a, NSAM computes the convex hull over the observed numeric state variables' values in every pre-state of every state transition involving a. To learn numeric effects, NSAM tries to fit a linear regression algorithm for each state variable and its post-state values based on all the pre-state variables' values. NSAM provides the same safety guarantee as SAM.

3 Problem Definition

In this work, we deal with the problem of learning an action model $\hat{M}$ for a numeric planning domain $D = \langle B, N, A, M \rangle$ given a set of trajectories $\mathcal{T}$. Our objective is to learn an action model to solve numeric planning problems in D. Thus, we do not necessarily aim for the action model we learn ($\hat{M}$) to be equal to the domain's real action model (M), but rather that $\hat{M}$ will *enable* solving planning problems in D when using a domain-independent planner. Formally, an action model $\hat{M}$ is said to enable solving a problem $\Pi = \langle D, s_0, G \rangle$ with a domain-independent planner P if calling P with input $\langle D_{\hat{M}}, s_0, G \rangle$ outputs a plan π for Π, where $D_{\hat{M}} = \left\langle B, N, A, \hat{M} \right\rangle$, i.e., the domain with action model $\hat{M}$ and the state variables and actions from D.

Assumptions We make the following assumptions about the domain D and the given set of trajectories $\mathcal{T}$. The preconditions in D are conjunctions of Boolean propositions and linear inequalities, and the effects in D are assignments over Boolean propositions and linear combinations of numeric functions. The trajectories in $\mathcal{T}$ are created by observing *successful executions* of actions in plans that solve problems in D. That is, our trajectories include only *positive observations*. Importantly, these input trajectories may, and often will, originate from different initial states and solve different problems in D. We also assume the state transitions in $\mathcal{T}$ are fully observable states and noise-free. Thus, we can consider $\mathcal{T}$ as an unordered set of state transitions. These assumptions are common in numeric planning and in the learning action model literature. Learning a numeric action model is still a very hard problem because learning numeric preconditions in our setting requires learning a *linear separator* using only *positive examples*, which is known to be hard. In our context, the linear separator is a linear precondition and the positive example is an observed transition (s, a, s'), indicating that the preconditions of action a are satisfied in state s. A *negative example* in our context is a negative observation, i.e., a pair (s, a) where s is a state where a is inapplicable in s according to the real action model M. As described above, we are not given access to negative observations.

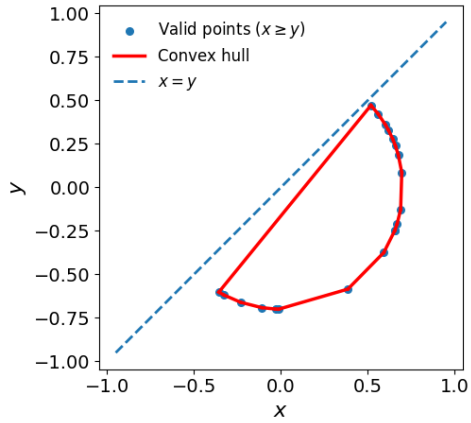

Figure 1: Example points of pre-states that follow the condition $x \geq y$. The dashed line represents the line of $x = y$ and the red lines are the lines created by the convex hull of the example points.

Hardness Juba, Le, and Stern (2021) showed that for classical planning domains, using only positive observations, the SAM algorithm has a PAC-like guarantee on the sample complexity. Namely, given a distribution $Dist$ over solvable planning problems in a domain D. If the SAM learning algorithm is given enough observations of an agent solving problems in $Dist$, then, with a high probability $\geq 1 - \delta$, it will learn an action model that only fails to solve a planning problem in $Dist$ with a probability $\leq \epsilon$. Consequently, given enough observation, the preconditions learned by SAM will only include propositions that, with high probability, *must* be part of the actions' preconditions.

Unfortunately, Mordoch, Juba, and Stern (2023) showed that the same does not apply when learning numeric planning action models with NSAM. Specifically, learning numeric preconditions that provide PAC-like guarantees only from positive observations while maintaining the safety property can be infeasible in some domains. Mordoch, Juba, and Stern (2023) illustrated this problem with the following example from Goldberg (1992). Consider a domain with two numeric functions. Let a be an action with the precondition $x \geq y$. Assume that the states where a can be applied on are distributed uniformly on a circle where the preconditions hold as presented in Figure 1. For any two observed points $(x_i, y_i), (x_j, y_j)$ on the circle, the convex shape composed of these two points is the straight line connecting them, thus,

NSAM will create the line connecting (x_i, y_i) and (x_j, y_j). In this case, NSAM cannot generalize past its observed points as any other point on the *arc* connecting $(x_i, y_i), (x_j, y_j)$ is deemed inapplicable according to the learned preconditions.

Despite these negative theoretical results, empirical results showed that NSAM can learn useful action models for some standard planning benchmarks (Mordoch, Juba, and Stern 2023). Yet NSAM failed to learn effective action models in other planning benchmarks. Our work aims to close this gap by relaxing NSAM's safety requirement and simulating possible negative observations.

4 SAM-SVM

Our main contribution in this work is the SAM-SVM algorithm for learning numeric action models. SAM-SVM uses SAM to learn the Boolean preconditions and effects. To learn the numeric preconditions, it first generates *simulated negative observations*, representing an informed hypothesis of states in which different actions cannot be applied. We propose below several heuristic methods for doing this. Then, SAM-SVM iteratively applies the Support Vector Machine (SVM) algorithm (Hearst et al. 1998) until obtaining a set of hyperplanes that completely separates the positive observations from the simulated negative observations. The conjunction of these hyperplanes defines the numeric preconditions. Finally, the numeric effects are learned using linear regression, as is done by NSAM.

Algorithm 1 provides a pseudo-code for SAM-SVM. First, SAM-SVM learns the Boolean preconditions and effects using the SAM algorithm (line 1). Then, for each action a, the algorithm initializes the datasets corresponding to the positive pre- and post-states ($\mathcal{P}^+_{pre}(a)$ and $\mathcal{P}^+_{post}(a)$), and the simulated negative pre-states ($\mathcal{P}^-_{pre}(a)$) (line 3). For each observed transition $\langle s, a, s' \rangle$, the algorithm adds s to $\mathcal{P}^+_{pre}(a)$ and s' to $\mathcal{P}^+_{post}(a)$ (lines 5–6). Next, the algorithm generates simulated negative examples using the selected heuristic (line 8). Using both the positive and simulated negative examples, SAM-SVM iteratively applies the SVM algorithm to infer the numeric preconditions (line 10). Finally, SAM-SVM learns the numeric effects by applying linear regression to the positive pre- and post-state examples.

4.1 Simulating Negative Observations

One may consider many different ways to simulate negative observations. In this work, we propose two heuristic methods for doing so called Expanded Convex Hull (ECH) and Not Used Observations (NUO). We describe these methods below in detail.

Expanded Convex Hull (ECH). The rationale for this heuristic method is that points lying some distance outside the convex hull of the positive observations are likely to be negative observations. Concretely, ECH computes the convex hull of the given positive observations like NSAM, and then *expands* this convex hull to all directions. Then, it selects random points on the facet of the expanded hull and designates them as negative examples.

Algorithm 1 The SAM-SVM Algorithm

Require: Observed transitions $\mathcal{T}$, Heuristic h
Ensure: Learned action model $\mathcal{M}$
1: $pre_{bool}, eff_{bool} \leftarrow \text{SAM}(\mathcal{T})$
2: **for all** $a \in A$ **do**
3: $\quad \mathcal{P}^+_{pre}(a), \mathcal{P}^+_{post}(a), \mathcal{P}^-_{pre}(a) \leftarrow \emptyset$
4: **for** $\langle s, a, s' \rangle$ appearing in $\mathcal{T}$ **do**
5: $\quad \mathcal{P}^+_{pre}(a) \leftarrow s$
6: $\quad \mathcal{P}^+_{post}(a) \leftarrow s'$
7: **for all** $a \in A$ **do**
8: $\quad P^-_{pre}(a) \leftarrow h\left(\mathcal{P}^+_{pre}(a)\right)$
9: $\quad$ **repeat**
10: $\quad\quad H_a \leftarrow \text{SVM}(\mathcal{P}^+(a), \mathcal{P}^-(a))$
11: $\quad\quad pre_{num}(a) \leftarrow pre_{num}(a) \cup H_a$
12: $\quad$ **until** convex hyperplanes are obtained
13: $\quad eff_{num}(a) \leftarrow \text{LINEARREGRESSION}(\mathcal{P}^+_a)$
14: $\mathcal{M} \leftarrow$ combine Boolean and numeric *pre* and *eff*
15: **return** $\mathcal{M}$

Algorithm 2 provides a more detailed description of how ECH works. ECH is a parametric method, accepting a parameter ε specifying how far away from the NSAM convex hull to position the simulated negative observations. If the epsilon is too small, the generated negative examples sit close to the boundary of the observed states, forcing the algorithm to learn overly restrictive models. Conversely, if epsilon is too large, the geometric boundaries become too loose, risking the generation of inapplicable plans. ECH starts by computing the convex hull $\mathcal{C}$ of the positive observations as is done by NSAM (line 3). Then, ECH calculates the centroid of the convex hull, c, by finding the mean of all the points composing $\mathcal{C}$ (line 4). Then, for each hyperplane $H \in \mathcal{C}$ and each vertex v of the hyperplane, ECH calculates its direction vector $d = v - c$ and expands the hyperplane outward as $v' = c + (1+\epsilon)d$, adding the resulting point to the expanded set E (lines 8-12). A new convex hull C_{exp} is then computed over the union of the original point and the expanded points (line 13). Finally, random points are sampled from the outer hyperplanes of C_{exp} to serve as negative examples N (lines 14-15).

Not Used Observations (NUO). The rationale for this heuristic method is to consider as negative observations every observed state in which the action was not executed, provided that it does not overlap with a positive example. In more detail, NUO works as follows. For a given action a, it iterates over all observed state transitions $\langle s, \hat{a}, s' \rangle$ and selects those such that $a \neq \hat{a}$. For each such transition, it constructs a vector consisting of the numeric state variables of s and checks whether this vector lies within the convex hull defined by the positive examples of a. If the vector is contained within the convex hull, the transition is discarded. Otherwise, the vector is selected as a negative example.

To illustrate the behavior of ECH and NUO, Fig. 2 plots the simulated negative observations and generated preconditions when using SAM-SVM with ECH (left plot) and with

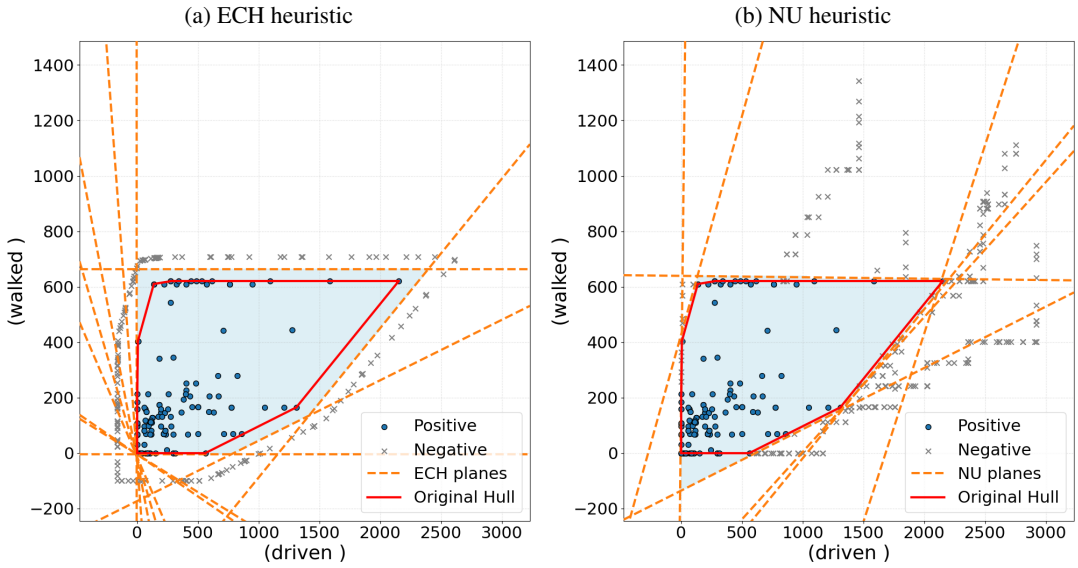

Figure 2: The convex conditions learned while using an SAM algorithm with the simulated negative points. The left figure represents conditions learned using the ECH heuristic with $\epsilon = 0.3$, and the right figure represents conditions learned using the NUO heuristic.

Algorithm 2 Negative Points via Expanded Convex Hull

input: Positive examples P, ε
output: Negative examples N
Compute convex hull $\mathcal{C}$ from P
Compute centroid c
$N \leftarrow \emptyset$
$E \leftarrow \emptyset$
for each hyperplane H of $\mathcal{C}$ **do**
    for each vertex $v \in H$ **do**
        $d \leftarrow v - c$
        $v' \leftarrow c + (1 + \varepsilon)\, d$
        Add v' to expanded set E
Compute convex hull $\mathcal{C}_{exp}$ from $P \cup E$
Sample random points S from hyperplanes $\mathcal{H}'$ of $\mathcal{C}_{exp}$
$N \leftarrow N \cup S$
return N

NUO (right plot). A key property of the ECH heuristic is that it can *always* generate negative observations, since the numeric state space is infinite and the convex hull can therefore be expanded indefinitely. In contrast, as additional observations are accumulated, the NUO heuristic more accurately identifies actions that do not contain numeric preconditions. This follows from the fact that only a small number of states will be classified as negative examples by the heuristic.

It is important to emphasize that both methods are heuristics and may yield inaccurate classifications. For instance, consider an action a that has no preconditions and is thus applicable in every state. By construction, the ECH heuristic will nevertheless generate negative examples based on the states in which a was executed. Similarly, if there exists another action $a' \neq a$ that is applied in states different from those in which a is executed, the NUO heuristic will also produce negative observations for a. We observed this phenomenon in our experiments on the Sailing domain (Scala, Haslum, and Thiébaux 2016), where all actions except one are defined without preconditions.

4.2 Handling Negative Preconditions

Most standard planning benchmark domains do not have negative (Boolean) preconditions, and indeed some planners do not support such preconditions. Nevertheless, SAM learns both positive and negative propositions as action preconditions. SAM does so even in domains that do not have negative preconditions, to maintain its safety guarantee. In particular, in domains where delete effects are not required as preconditions, omitting negative preconditions can lead to discrepancies between the learned and true action models. In such cases, if a delete effect is unobserved, executing the action under the learned model may yield a successor state that differs from that produced by the true action model.

We observed that this behavior of SAM severely limits its ability to learn useful action models in some domains. For example, in the DRIVERLOG domain, for the `drive-truck` action, SAM learned six additional negative preconditions. Similarly, for the `walk` action, SAM learned three additional negative preconditions. Therefore, we modified how SAM-SVM learns Boolean preconditions by removing any negative preconditions created by SAM. This increased action applicability of the learned action model, at the cost of losing SAM's safety guarantee in some domains. We made this design choice since SAM-SVM anyhow does not provide strict safety guarantees due to its use of the simulated negative observations.

4.3 Interleaving Safe and Unsafe Action Models

As discussed above, SAM-SVM with either negative observation method loses the NSAM safety guarantee. Indeed, the

preconditions generated by SAM-SVM may be "too loose", i.e., planning with the action model it generates may output a plan that includes executing actions in states where they are inapplicable according to the real action model. To mitigate this limitation, we suggest integrating SAM-SVM and NSAM as follows. Given the input set of trajectories $\mathcal{T}$, we run both SAM-SVM and NSAM, obtaining two action models $\hat{M}$ and M_{safe}, respectively. For each planning instance, we first try solving it using the action model generated by the NSAM algorithm (M_{safe}). If a plan was found, we return it, as it is guaranteed to be applicable in the real domain. If no plan was found with M_{safe}, we attempt to solve the given problem with SAM-SVM. We call this hybrid of NSAM and SAM-SVM as NSAM+SVM.

Using NSAM+SVM has pros and cons. The benefit of NSAM+SVM over SAM-SVM is that it is guaranteed to be able to solve at least as much, and possibly more, problems than SAM-SVM, due to the safety property of NSAM. On the other hand, using NSAM+SVM potentially requires double the planning time, as it requires planning with both models. Nevertheless, one may parallelize these planner calls so that overall CPU time remains manageable.

4.4 Runtime Analysis

The runtime of SAM-SVM is composed of: (1) learning the Boolean action model using the SAM algorithm; (2) constructing the pre- and post-state numeric datasets; (3) generating simulated negative observations using either ECH or NUO; (4) performing iterative SVM calls to derive the numeric preconditions; and (5) applying linear regression to estimate the numeric effects.

The runtime of SAM is linear in the number of state transitions, actions, and propositions, namely $O(|T| \cdot |A| \cdot |B|)$, where $|T|$ denotes the number of observed transitions, $|A|$ the number of actions, and $|B|$ the number of Boolean propositions. Constructing the pre- and post-state numeric datasets requires iterating over all transitions and updating $\mathcal{P}^{+}_{pre}(a)$ and $\mathcal{P}^{+}_{post}(a)$ for each action a, yielding a time complexity of $O(|T| \cdot |A| \cdot |N|)$, where $|N|$ is the number of numeric functions.

Both ECH and NUO require computing the convex hull over the positive examples for each action. We compute the convex hull using the Quickhull algorithm (Barber, Dobkin, and Huhdanpaa 1996), as in NSAM. The complexity of convex hull computation depends on the dimensionality, which is bounded by $|N|$. Assuming $|N| \geq 4$, the runtime is $O\left(\frac{|T|^{\lfloor |N|/2 \rfloor}}{\lfloor |N|/2 \rfloor!}\right)$. In addition, ECH incurs extra overhead for constructing the expanded convex hull used to generate negative examples.

The NUO heuristic inserts each observed state into the negative candidate sets of all non-executed actions, resulting in $O(|T| \cdot |A| \cdot |N|)$ time. After computing the convex hull of the positive examples, candidate negatives that lie inside the hull are filtered out by verifying compliance with the hyperplane equations, which requires $O(|A| \cdot |T|^{\lfloor |N|/2 \rfloor})$ time.

The SVM learning phase is implemented using LIBLINEAR (Fan et al. 2008), whose complexity is $O(|T| \cdot |N|)$ per action. In the worst case, where a separate linear separator is constructed for each negative example, the total complexity becomes $O(|A| \cdot |T|^2 \cdot |N|)$.

Estimating numeric effects via least-squares linear regression is equivalent to solving $(X^\top X)^{-1} X^\top y$ (Watson 1967). For each potential numeric effect, this yields a worst-case runtime of $O(|N|^2 \cdot (|T| + |N|))$. Consequently, computing the numeric effects for all actions results in a total complexity of $O(|A| \cdot |N|^3 \cdot (|T| + |N|))$. Finally, the overall runtime complexity of the SAM-SVM algorithm when using the NUO heuristic is:

$$\begin{aligned}\mathcal{T}_{\text{NUO}} = {} & O(|T| \cdot |A| \cdot |B|) \\ & + O(|T| \cdot |A| \cdot |N|) \\ & + O\left(|A| \cdot \frac{|T|^{\lfloor |N|/2 \rfloor}}{\lfloor |N|/2 \rfloor!}\right) \\ & + O\left(|A| \cdot |T|^{\lfloor |N|/2 \rfloor}\right) \\ & + O(|A| \cdot |T|^2 \cdot |N|) \\ & + O\big(|A| \cdot |N|^3 \cdot (|T| + |N|)\big)\end{aligned}$$

When using the ECH heuristic the runtime complexity is:

$$\begin{aligned}\mathcal{T}_{\text{ECH}} = {} & O(|T| \cdot |A| \cdot |B|) \\ & + O(|T| \cdot |A| \cdot |N|) \\ & + O\left(|A| \cdot \frac{|T|^{\lfloor |N|/2 \rfloor}}{\lfloor |N|/2 \rfloor!}\right) \\ & + O(|A| \cdot |T|^2 \cdot |N|) \\ & + O\big(|A| \cdot |N|^3 \cdot (|T| + |N|)\big)\end{aligned}$$

For both heuristics, the runtime complexity is exponential in $|N|$, polynomial in $|T|$ and linear in $|A|$ and $|B|$.

Domain	$\#a$	$\#a^N$	$\#B$	$\#N$
Counters	4	4	0	3
Depot	5	4	6	4
Driverlog	6	2	6	4
Farmland	2	2	1	2
Minecraft	6	6	1	6
Sailing	8	8	1	3
Satellite	5	2	8	6
Rovers	10	9	26	2

Table 1: The properties of the experimented domains.

5 Experimental Results

We evaluated our methods on numeric domains that satisfy two conditions: (1) a problem generator exists for each domain, (2) the numeric preconditions and effects are linear, and (3) at least some actions include numeric preconditions and effects. Specifically, we experimented on the domains listed in Table 1. Driverlog, Depots, Rovers, and Satellite were taken from the 3rd International Planning Competition (IPC3) (Long and Fox 2003), Farmland and Sailing from (Scala, Haslum, and Thiébaux 2016), Counters

Traj.	Baselines		SAM-SVM					NSAM+SVM				
	NSAM	PMnr	NUO	ECH (0.01)	ECH (0.1)	ECH (0.2)	ECH (0.3)	NUO	ECH (0.01)	ECH (0.1)	ECH (0.2)	ECH (0.3)
Counters												
10	0.07	0.52	**0.83**	0.08	0.12	0.11	0.12	0.82	0.10	0.15	0.15	0.15
20	0.42	0.56	0.83	0.42	0.42	0.45	0.47	**0.84**	0.43	0.44	0.47	0.50
30	0.70	0.56	0.82	0.68	0.68	0.67	0.67	**0.87**	0.70	0.71	0.74	0.75
40	0.81	0.54	0.84	0.81	0.77	0.78	0.72	**0.89**	0.83	0.85	0.89	0.85
50	0.80	0.57	0.85	0.81	0.77	0.78	0.72	**0.87**	0.81	0.83	0.86	**0.87**
Farmland												
10	0.83	0.14	0.41	0.83	0.83	0.83	0.84	**0.86**	0.83	0.83	0.83	0.84
20	0.95	0.13	0.42	0.95	0.95	0.94	0.95	**0.96**	0.95	0.95	**0.96**	**0.96**
30	0.93	0.12	0.34	0.92	0.94	0.95	0.94	0.95	0.93	0.94	**0.96**	**0.96**
40	0.96	0.12	0.27	0.95	0.95	0.96	0.96	**0.97**	0.96	0.96	**0.97**	**0.97**
50	0.95	0.12	0.26	0.94	0.93	0.96	0.92	0.96	0.95	0.95	**0.98**	**0.98**
Minecraft												
10	0.04	N/A	0.35	0.15	0.05	0.05	0.05	**0.36**	0.15	0.05	0.05	0.05
20	0.05	N/A	0.46	0.07	0.06	0.06	0.08	**0.46**	0.08	0.07	0.07	0.08
30	0.13	N/A	**0.61**	0.20	0.16	0.16	0.19	**0.61**	0.21	0.18	0.18	0.18
40	0.14	N/A	**0.62**	0.25	0.21	0.25	0.28	**0.62**	0.26	0.22	0.25	0.25
50	0.12	N/A	**0.65**	0.31	0.23	0.27	0.29	**0.65**	0.31	0.25	0.27	0.27
Sailing												
10	0.85	0.00	0.00	0.35	0.34	0.08	0.12	0.85	0.86	**0.90**	0.85	0.86
20	**0.90**	0.00	0.00	0.00	0.00	0.00	0.01	**0.90**	**0.90**	**0.90**	**0.90**	**0.90**
30	**0.94**	0.00	0.00	0.00	0.01	0.00	0.00	**0.94**	**0.94**	**0.94**	**0.94**	**0.94**
40	**0.97**	0.00	0.00	0.01	0.02	0.00	0.00	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**
50	**0.97**	0.00	0.00	0.02	0.00	0.00	0.00	0.96	0.96	0.96	0.96	0.96

Table 2: Problem-solving rates for Minecraft, Counters, Sailing, and Farmland. Best results per configuration is marked in bold.

Traj.	Baselines		SAM-SVM					NSAM+SVM				
	NSAM	PMnr	NUO	ECH (0.01)	ECH (0.1)	ECH (0.2)	ECH (0.3)	NUO	ECH (0.01)	ECH (0.1)	ECH (0.2)	ECH (0.3)
Rovers												
10	0.19	0.17	0.38	0.41	0.43	0.44	**0.46**	0.38	0.40	0.43	0.44	**0.46**
20	0.37	0.61	0.56	0.71	0.71	**0.72**	0.65	0.60	0.70	0.71	**0.72**	0.65
30	0.45	0.49	0.57	**0.75**	0.73	0.73	0.65	0.66	**0.75**	0.73	**0.75**	0.69
40	0.47	0.45	0.61	0.78	**0.79**	0.73	0.73	0.69	**0.79**	**0.79**	0.76	0.75
50	0.49	0.54	0.62	**0.81**	0.80	0.73	0.72	0.69	**0.81**	0.80	0.75	0.73
Depots												
10	0.27	**0.96**	0.94	0.57	0.40	0.59	0.63	0.93	0.57	0.42	0.58	0.62
20	0.49	**0.96**	0.93	0.68	0.58	0.72	0.77	0.93	0.68	0.60	0.72	0.79
30	0.55	**0.96**	0.92	0.71	0.59	0.75	0.70	0.93	0.70	0.62	0.77	0.71
40	0.59	**0.96**	0.93	0.72	0.67	0.75	0.81	0.94	0.71	0.69	0.75	0.81
50	0.66	**0.96**	0.94	0.79	0.76	0.80	0.83	**0.96**	0.79	0.75	0.79	0.81
Driverlog												
10	0.47	**1.00**	0.70	0.63	0.71	0.72	0.63	0.73	0.66	0.71	0.72	0.66
20	0.60	**1.00**	0.74	0.58	0.70	0.68	0.68	0.75	0.67	0.74	0.69	0.70
30	0.58	**1.00**	0.70	0.66	0.66	0.72	0.68	0.72	0.70	0.68	0.73	0.68
40	0.58	**1.00**	0.73	0.62	0.66	0.66	0.68	0.74	0.67	0.64	0.70	0.69
50	0.57	**1.00**	0.75	0.63	0.69	0.69	0.67	0.72	0.68	0.70	0.68	0.68
Satellite												
10	0.03	**0.89**	0.29	0.13	0.13	0.03	0.07	0.30	0.15	0.13	0.05	0.07
20	0.08	**0.91**	0.28	0.27	0.20	0.29	0.28	0.31	0.24	0.19	0.23	0.26
30	0.05	**0.91**	0.32	0.28	0.30	0.32	0.29	0.32	0.26	0.27	0.31	0.28
40	0.06	**0.91**	0.35	0.29	0.34	0.33	0.41	0.34	0.25	0.31	0.32	0.38
50	0.06	**0.91**	0.37	0.32	0.32	0.31	0.34	0.39	0.30	0.28	0.31	0.33

Table 3: Problem-solving rates for Rovers, Depots, Driverlog, and Satellite. The best result per configuration is marked in bold.

from IPC 2023 (Scala et al. 2020; Taitler et al. 2024), and Minecraft from Benyamin et al. (2023). Table 1 provides general statistics for each domain. The columns $\#a$ and $\#a^N$ denote the number of (lifted) actions and the subset of actions containing numeric preconditions or effects, respectively. The columns $\#B$ and $\#N$ represent the number of Boolean and numeric variables, respectively.

Trajectory generation. First, we generated planning problems for each domain using the problem generator in github.com/SPL-BGU/pddl-problem-generator. To solve the generated planning problems, we used a portfolio of state-of-the-art numeric planners, namely ENHSP (Scala, Haslum, and Thiébaux 2016) and Metric-FF (Hoffmann 2003).[1] Both planners expose several configuration options and parameters, including modes for optimal and satisfiable planning. We are interested in the satisfiable configuration as our main objective is to learn an action model that allows solving problems, rather than optimizing their cost. Specifically, ENHSP was executed using `sat-hmrphj` configuration, which employs Greedy Best-First Search with the MRP heuristic, helpful actions, and helpful transitions. Metric-FF was executed using its standard configuration, which applies Enforced Hill Climbing (EHC) followed by BFS. Both planners were configured to use a numeric tolerance threshold of 0.1 in their calculations and a timeout of 10 minutes. From the solved instances, we sampled 100 problems for each domain and split them into 80% training and 20% test sets. From the training portion, we constructed datasets of sizes ranging from 10 to 50 trajectories (one trajectory per solved planning problem). We performed five-fold cross-validation on the training data. Results were averaged across all folds to reduce variance and improve statistical reliability. All experiments were performed on a SLURM CPU cluster with a 32 GB RAM limit and 32 GB of storage allocated per job.

Algorithms. We implemented SAM-SVM with the two heuristic methods for simulating negative observations, ECH and NUO, denoted SAM-ECH and SAM-NUO , respectively. For ECH, we experimented with $\varepsilon \in \{0.01, 0.1, 0.2, 0.3\}$. We also implemented the hybrid NSAM+SVM algorithm, which combines NSAM and SAM-SVM. NSAM+SVM with ECH is denoted by NSAM-ECH and NSAM+SVM with NUO is denoted by NSAM-NUO . As baselines, we compared our algorithms to PMnr (Segura-Muros, Pérez, and Fernández-Olivares 2021) and NSAM, which are the only numeric ac-

[1]We slightly modified Metric FF to better control numeric imprecision issues.

	Domain							
Algorithm	**Counters**	**Depots**	**Driverlog**	**Farmland**	**Sailing**	**Satellite**	**Rovers**	**Minecraft**
NSAM	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.13
PMnr	0.37	0.04	0.00	0.73	1.00	0.00	0.45	N/A
SAM-NUO	0.00	0.02	0.00	0.73	1.00	0.35	0.35	0.05
NSAM-NUO	0.00	0.01	0.00	0.04	0.04	0.33	0.28	0.05
SAM-ECH (0.01)	0.00	0.00	0.00	0.00	0.54	0.02	0.00	0.00
NSAM-ECH (0.01)	0.00	0.00	0.00	0.00	0.01	0.02	0.00	0.00
SAM-ECH (0.1)	0.04	0.00	0.00	0.00	0.84	0.01	0.02	0.00
NSAM-ECH (0.1)	0.00	0.00	0.00	0.00	0.02	0.01	0.02	0.00
SAM-ECH (0.2)	0.08	0.00	0.00	0.00	0.84	0.02	0.08	0.00
NSAM-ECH (0.2)	0.00	0.00	0.00	0.00	0.02	0.02	0.06	0.00
SAM-ECH (0.3)	0.16	0.00	0.00	0.04	0.83	0.05	0.12	0.03
NSAM-ECH (0.3)	0.00	0.00	0.00	0.01	0.02	0.02	0.11	0.02

Table 4: Average false-plan ratios across experimental domains for the evaluated algorithms using 50 trajectories.

	Domain															
Algorithm	**Counters**		**Depots**		**Driverlog**		**Farmland**		**Sailing**		**Satellite**		**Rovers**		**Minecraft**	
	# Pre.	Time	# Pre.	Time	# Pre.	Time	# Pre.	Time	# Pre.	Time	# Pre.	Time	# Pre.	Time	# Pre.	Time
NSAM	51.40	8.35	808.40	6.39	180.40	9.38	15.20	1.22	94.60	10.04	3307.60	15.56	52.00	2.97	46.60	2.68
PMnr	2.00	15.84	0.00	35.25	0.00	217.40	0.73	5.49	1.20	18.91	3.00	56.72	0.00	23.39	N/A	N/A
SAM-ECH (0.01)	54.40	7.79	132.00	32.07	82.60	15.99	16.20	0.90	95.80	13.09	198.80	263.35	55.00	4.47	38.80	3.29
SAM-ECH (0.1)	46.40	7.14	292.00	12.34	109.20	10.19	18.00	0.94	62.40	9.16	520.00	153.88	56.60	3.44	50.40	2.71
SAM-ECH (0.2)	36.20	9.01	170.00	7.38	82.60	9.20	15.60	0.91	49.60	11.97	425.80	53.22	51.60	3.30	55.20	2.05
SAM-ECH (0.3)	33.20	6.85	132.00	7.80	76.20	12.04	14.20	0.91	45.60	10.14	292.00	36.79	49.40	3.69	53.20	2.08
SAM-NUO	29.40	15.34	3.40	27.99	28.20	19.31	3.20	1.21	56.20	28.99	148.40	23.33	6.00	13.94	22.00	5.33

Table 5: Average number of learned numeric preconditions (column "#Pre.") and learning runtime in seconds ("Time") across different planning domains for the evaluated algorithms using 50 trajectories.

tion model learning algorithms prior to our work. Since PMnr runtime is potentially unbounded, we restricted its runtime to 30 minutes. If the execution does not finish within 30 minutes we terminate its execution and assume that no action model was learned.

Evaluation Metrics. We evaluated the learned models using the *problem-solving ratio* and the *false-plan ratio metrics* defined by (Stern et al. 2025). These metrics are computed with respect to a set of test problems T_{test}. The problem-solving ratio is the ratio of problems in T_{test} that were solved and validated within a 10 minutes time limit. To evaluate the NSAM+SVM hybrid in a fair way, we split this 10 minutes time limit in half, allocating the first 5 minutes to planning with the safe action model returned by NSAM and the next 5 minutes to planning with the action model returned by SAM-SVM. We validated every generated plan against the real action model using VAL (Howey, Long, and Fox 2004). The false-plan ratio is the ratio of problems that were solved using the learned action model but VAL identified the plan is inapplicable in the real action model. We configured VAL to support the same numeric tolerance as configured for the planner (0.1).

5.1 Results: Problem-Solving Ratio

Tables 2 and 3 report the problem-solving rates as a function of the number of input trajectories across the evaluated domains. Column `Traj.` denotes the number of input trajectories used for learning the action model. The remaining columns correspond to the evaluated algorithms for each domain. Specifically, we compare the baseline algorithms NSAM and PMnr with SAM-NUO , SAM-ECH for $\varepsilon \in \{0.01, 0.1, 0.2, 0.3\}$, NSAM-NUO , and NSAM-ECH with the corresponding ε values. Observe that in some cases learning from more trajectories reduces the problem-solving rate. While this is not common in our results, it may occur because learning from more trajectories may yield a more accurate action model that is also more complex. In some cases, this increased complexity leads to more timeouts and thus a lower problem-solving ratio.

Overall, NSAM-NUO yields the most robust performance in most domains, achieving either the best results or very close to it in 5 out of 8 domains. For example, in the FARMLAND domain, NSAM-NUO achieved the highest success rates using 10, 20, and 40 input trajectories (86%, 96%, and 97%, respectively). A similar trend is observed in SAILING; however, since SAM-NUO fails to solve any instances in this domain, the observed improvement is attributed to the un-

derlying NSAM component. In general, there is no universal winner that outperforms all other algorithms across all domains and configurations. Thus, we investigate our results across the following axes.

Comparison between ECH and NUO in SAM-SVM. The SAM-NUO achieves results comparable or better than the SAM-ECH configurations in the COUNTERS, DEPOTS, DRIVERLOG, MINECRAFT, and SATELLITE domains. For example, in DEPOTS SAM-NUO exceeds the performance of all SAM-ECH instances by more than 10%, with similar trends observed in the other domains. Yet in FARMLAND and ROVERS, SAM-ECH attains higher solving rates. We observed that this is due to NUO not generating sufficient simulated negative observations in these domains. By contrast, ECH generates additional simulated negative observations around all positive observations, leading in this case to more accurate precondition approximations.

Comparison with NSAM. Across all domains except FARMLAND and SAILING, SAM-NUO consistently outperforms the NSAM baseline. In SAILING domain, only the *save-person* action has numeric preconditions, which are strict. Approximating these preconditions – either via SAM-NUO or SAM-ECH renders most plans inapplicable. In FARMLAND, trajectories mostly consist of the *move-slow* action, while *move-fast*—the only other action—is rarely observed. As a result, simulating negative observations using the NUO heuristic for the action *move-slow* is infeasible. Thus, the numeric preconditions are not learned, rendering the action inapplicable. Although SAM-NUO successfully learns numeric preconditions for *move-fast*, this action is seldom used, causing most generated plans to be invalid.

Comparison with PMnr. The baseline PMnr presents the worst performance in the COUNTERS, FARMLAND, MINECRAFT, and SAILING domains, while it performed well in the DEPOTS, DRIVERLOG, and SATELLITE domains, achieving solving rates of 96%, 100% and 91%, respectively. This behavior can be attributed to the number of numeric preconditions in each domain. In domains with very few numeric preconditions, such as DEPOTS, DRIVERLOG, and SATELLITE, PMnr works well, but performs poorly in domains with more numeric preconditions.[2] The low solving rates of PMnr in COUNTERS, FARMLAND, and SAILING are primarily due to a large fraction of inapplicable plans (more details on this later in Table 4).

Comparison of SAM-SVM and NSAM+SVM. When comparing SAM-SVM and NSAM+SVM it is clear that, as expected, in almost all cases using NSAM+SVM yields much better results than SAM-SVM. For example, in SAILING SAM-SVM fails to solve most problems while NSAM+SVM utilizes NSAM to solve most problems in this domain. We observe that in some domains, NSAM+SVM achieved slightly lower problem-solving rates than the NSAM baseline. For example, in SAILING with 50 input trajectories NSAM obtains a 97% success rate while NSAM-NUO achieved only 96%. We attribute this discrepancy to the reduced per-instance timeout used in these experiments, which prevented some problems from being solved.

[2] In MINECRAFT, PMnr fails entirely, as it assumes actions have parameters and crashes when this assumption is violated.

5.2 Results: False-Plan Rates

Table 4 reports on the false-plans rates obtained after learning from 50 input trajectories across all domains. Among all approaches, PMnr exhibits the highest false-plan rates in every domain except SATELLITE. The second-highest rates are observed for SAM-NUO , which can be explained by an insufficient number of simulated negative observations to induce accurate numeric preconditions. This effect is particularly pronounced in the FARMLAND and ROVERS domains.

As expected, the hybrid NSAM+SVM, which integrates NSAM with the unsafe SAM-SVM, substantially reduces false-plan rates. For instance, in the SAILING domain, SAM-ECH and $\epsilon = 0.01$ yield 54% inapplicable plans; when integrated with NSAM (i.e., NSAM-ECH), this rate drops to 1%. A similar effect is observed in the ROVERS domain for SAM-NUO , where the hybrid NSAM-NUO reduced the false-plan rate from 35% to 28%.

Since NSAM returns a safe action model, by definition, it never outputs a false plan. An exception to this occurred in the MINECRAFT domain, where NSAM obtained a 13% false-plan ratio. This occurred due to numeric imprecision and the tolerances to them configured to the planner. Importantly, even though we configured VAL to permit numeric imprecision, the numeric error of the planners accumulated and resulted in invalid plans. Furthermore, when we tested the generated plans against the action model learned by NSAM we observed that the plans were not applicable against the learned action model as well. Interestingly, the NSAM+SVM algorithm sometimes yielded lower false-plan rates than NSAM in MINECRAFT. We attribute this to the fact that some plans classified as inapplicable by the safe model are successfully executed by the unsafe learner.

5.3 Results: Runtime and Complexity

Table 5 reports for each domain the number of learned numeric preconditions (column "# Pre.") and learning runtime in seconds (column "Time") after 50 learning trajectories. As can be seen, the runtime of NSAM across all domains is below 16 seconds on average. For SAM-SVM with the NUO heuristic, the runtime increased to at most 29 seconds. For SAM-SVM with the ECH heuristic, we observe that in general the runtime increases as ϵ decreases. The runtime remained manageable – less than 33 seconds on average – except for *Satellite*, where the average runtime was 36, 53, 153, and 263 seconds for ϵ values of 0.3, 0.2, 0.1, and 0.01, respectively.

Next, consider the complexity of the learned models, as measured by the number of numeric preconditions (Column "#Pre"). The results show that in most cases NUO returns simpler models than ECH, and ECH yields simpler models than NSAM. For instance, in the Satellite domain, NUO learns an average of 148 numeric preconditions, compared to a minimum of 198 preconditions for ECH (with $\epsilon = 0.01$), and up to 3,307 preconditions for NSAM.

In general, in domains containing many numeric preconditions, NSAM+SVM outperforms the baseline approaches.

In domains with a small number of actions where a single action is executed predominantly – such as Farmland – the ECH heuristic outperforms NUO. In other domains, such as Counters, Minecraft, Depots, Driverlog, and Satellite, the NUO heuristic outperforms ECH. Conversely, in domains with few or no numeric preconditions, PMnr achieves superior performance compared to all other methods, as it correctly infers the absence of numeric preconditions.

6 Conclusion and Future Work

In this work, we presented SAM-SVM, a novel approach for learning numeric action models from positive observations. SAM-SVM is built on NSAM but relaxes its safety requirements by simulating negative observations. It iteratively uses SVM to learn linear separators between the positive observations and the simulated negative observations. We introduced two heuristics for generating simulated negative examples: ECH and NUO. The ECH heuristic samples negative points on hyperplanes obtained by expanding the convex hull that covers the positive observations, while the NUO heuristic treats states in which an action was not executed as negative examples. We also propose a hybrid NSAM+SVM algorithm that uses in tandem the action models learned by NSAM and by SAM-SVM. We evaluated our algorithms across a variety of planning domains. The results demonstrate that our approach outperforms the baselines NSAM and PlanMiner in several domains, and that the hybrid NSAM+SVM algorithm further improves performance.

As future work, we plan to investigate alternative methods for generating negative examples that do not rely on the NSAM algorithm, such as approaches based on neural networks. Additionally, we intend to extend our framework to learning action models in settings where the input trajectories are noisy or partially observable, for example, when observations are provided in the form of images. Also in this work, we focused on learning action models that enable successful plan generation, without explicitly considering the quality of the resulting plans. An interesting direction for future research is to investigate the impact of trajectory quality, particularly trajectories generated from optimal plans, on the learning process and the ability of the resulting model to produce higher-quality solutions.

AI Declaration

The authors have not employed any Generative AI tools beyond checking for grammar mistakes and typos.

Acknowledgments

This research was partly supported by the Israel Science Foundation (ISF) grant #1238/23 awarded to Roni Stern, and by the Magneton Program of the Israel Innovation Authority (Grant No. 90011).

References

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

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

Arora, A.; Fiorino, H.; Pellier, D.; Etivier, M.; and Pesty, S. 2018. A review of learning planning action models. *Knowledge Engineering Review* 33.

Barber, C. B.; Dobkin, D. P.; and Huhdanpaa, H. 1996. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)* 22(4):469–483.

Benyamin, Y.; Mordoch, A.; Shperberg, S. S.; and Stern, R. 2023. Model learning to solve minecraft tasks. In *PRL Workshop in the International Conference on Automated Planning and Scheduling (ICAPS)*.

Bolander, T., and Gierasimczuk, N. 2018. Learning to act: qualitative learning of deterministic action models. *Journal of Logic and Computation* 28(2):337–365.

Cresswell, S., and Gregory, P. 2011. Generalised domain model acquisition from action traces. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 42–49.

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

Fan, R.-E.; Chang, K.-W.; Hsieh, C.-J.; Wang, X.-R.; and Lin, C.-J. 2008. Liblinear: A library for large linear classification. *the Journal of machine Learning research* 9:1871–1874.

Fox, M., and Long, D. 2003. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Goldberg, P. W. 1992. *PAC-learning geometrical figures*. Ph.D. Dissertation, University of Edinburgh.

Gösgens, J.; Jansen, N.; and Geffner, H. 2025. Learning lifted strips models from action traces alone: A simple, general, and scalable solution. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 189–197.

Gregory, P., and Lindsay, A. 2016. Domain model acquisition in domains with action costs. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 149–157.

Hearst, M. A.; Dumais, S. T.; Osuna, E.; Platt, J.; and Scholkopf, B. 1998. Support vector machines. *IEEE Intelligent Systems and their applications* 13(4):18–28.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J. 2001. Ff: The fast-forward planning system. *AI magazine* 22(3):57–57.

Hoffmann, J. 2003. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Howey, R.; Long, D.; and Fox, M. 2004. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence*, 294–301. IEEE.

Juba, B., and Stern, R. 2022. Learning probably approximately complete and safe action models for stochastic worlds. In *AAAI Conference on Artificial Intelligence*.

Juba, B.; Le, H. S.; and Stern, R. 2021. Safe learning of lifted action models. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 379–389.

Le, H. S.; Juba, B.; and Stern, R. 2024. Learning safe action models with partial observability. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 20159–20167.

Li, D.; Scala, E.; Haslum, P.; and Bogomolov, S. 2018. Effect-abstraction based relaxation for linear numeric planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 4787–4793.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research* 20:1–59.

Mordoch, A.; Scala, E.; Stern, R.; and Juba, B. 2024. Safe learning of PDDL domains with conditional effects. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 34, 387–395.

Mordoch, A.; Juba, B.; and Stern, R. 2023. Learning safe numeric action models. In *AAAI*, 12079–12086. AAAI Press.

Mourao, K.; Zettlemoyer, L.; Petrick, R.; and Steedman, M. 2012. Learning strips operators from noisy and incomplete observations. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 614–623.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramirez, M. 2016. Interval-based relaxation for general numeric planning. In *European Conference on Artificial Intelligence (ECAI)*. 655–663.

Scala, E.; Haslum, P.; Magazzeni, D.; Thiébaux, S.; et al. 2017. Landmarks for numeric planning problems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 4384–4390.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramirez, M. 2020. Subgoaling techniques for satisficing and optimal numeric planning. *Journal of Artificial Intelligence Research* 68:691–752.

Scala, E.; Haslum, P.; and Thiébaux, S. 2016. Heuristics for numeric planning via subgoaling. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 3228–3234.

Segura-Muros, J. Á.; Pérez, R.; and Fernández-Olivares, J. 2021. Discovering relational and numerical expressions from plan traces for learning action models. *Applied Intelligence* 1–17.

Stern, R., and Juba, B. 2017. Efficient, safe, and probably approximately complete learning of action models. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 4405–4411.

Stern, R.; Lamanna, L.; Mordoch, A.; Benyamin, Y.; Lauer, P.; Juba, B.; Behnke, G.; Muise, C.; Bercher, P.; Vallati, M.; Xi, K.; Wattad, O.; and Eliyahu, O. 2025. Evaluating planning model learning algorithms. In *Workshop on Knowledge Engineering for Planning and Scheduling (KEPS) at the International Conference on Automated Planning and Scheduling (ICAPS)*.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fišer, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; et al. 2024. The 2023 International Planning Competition.

Watson, G. S. 1967. Linear least squares regression. *The Annals of Mathematical Statistics* 1679–1699.