Safe Learning of Multi-Agent Action Models from Concurrent Observations

Anonymous submission

Abstract

Planning for a team of agents without knowing the agents' action model, i.e., the preconditions and effects of their actions, is difficult and error-prone. We present algorithms for learning agents' action models by observing these agents execute plans in the same domain. Learning agents' action models is particularly challenging when the observed agents can act concurrently since it requires attributing effects to individual agents' actions. We introduce the Multi-Agent Safe Action Model Learning (MA-SAM) algorithm to address this challenge. MA-SAM runs in polynomial time and the action model it returns is safe in the sense that plans generated with it are guaranteed to succeed. Then, we identify a limitation in MA-SAM that can yield unbounded sample complexity and introduce MA-SAM$^+$, which addresses this limitation while preserving the safety property. To demonstrate the applicability of our approach, we implemented MA-SAM and conducted experiments on a benchmark of Multi-Agent STRIPS domains. Despite the worst-case sample complexity, the action models returned by MA-SAM effectively solve more than 80% of the test set problems in most of the domains with fewer than 20 trajectories.

Introduction

Many real-world applications require planning for multiple agents, e.g., in furniture assembly systems (Knepper et al. 2013), automated warehouses (Azadeh, De Koster, and Roy 2019), sensor networks (Lesser, Ortiz Jr, and Tambe 2003), and search & rescue coordination (Allouche and Boukhtouta 2010). Planning for multiple agents often requires an action model, i.e., a formal description of the preconditions and effects of the agents' actions. Multiple formalisms have been proposed to represent and plan with such action models, such as MA-STRIPS (Brafman and Domshlak 2013) and Dec-POMDP (Bernstein et al. 2002; Tambe 1997).

However, such action models are often not available, e.g., in ad-hoc teamwork setups (Barrett and Stone 2012) where agents' interfaces may not support sharing their internal models (Verma, Marpally, and Srivastava 2021), or where agents do not wish to share this information due to privacy reasons (Brafman and Domshlak 2013; Maliah, Shani, and Stern 2016), or just because creating them manually is too difficult. In this work, we explore the problem of automatically learning an action model for multiple agents by observing the behavior of these agents in the domain. In particular, we consider the case where the observed actions may be carried out concurrently, which means the observed trajectories consist of joint actions. This poses a challenge in identifying the specific actions responsible for the effects observed in the states.

To address the challenge of learning an action model from trajectories of joint actions, we build on the Safe Action Model Learning (SAM) family of algorithms (Stern and Juba 2017; Juba, Le, and Stern 2021; Juba and Stern 2022; Mordoch, Juba, and Stern 2023). These algorithms learn planning action models from trajectories and guarantee that plans generated with the learned action model are safe to use. Algorithms from this family can support lifted action models (2021), action models with stochastic effects (2022), and action models with numeric preconditions and effects (2023). However, none of the SAM framework algorithms can learn action models in a concurrent multi-agent setting.

In this work, we bridge this gap and propose MA-SAM, a generalization of SAM learning (2021) that learns single-agent action models by observing trajectories where the agents act in parallel. We focus on a setting where agents act independently in the environment, a common assumption in MA-STRIPS (2008). Learning is still challenging in such settings, due to the ambiguity of effects. We prove that by contrast, the most challenging settings for multi-agent planning, where joint actions may have arbitrary, unrelated preconditions and effects (Boutilier and Brafman 2001), is essentially uninteresting from a learning standpoint: simply treating every joint action as a separate action is optimal, but requires a prohibitive number of observations. Learning is of interest primarily when some structure or regularity can be leveraged to generalize beyond a small number of observations.

In some cases, MA-SAM cannot discern which action created a specific effect. To resolve such ambiguities, we propose an extension to MA-SAM, named MA-SAM$^+$, to learn multi-agent macro actions. We theoretically an-

alyze these new algorithms, showing they preserve the safety property. We show that the above assumptions are necessary to obtain a tractable bound on the sample complexity required to learn a sufficiently powerful safe action model for each agent. We implemented MA-SAM and evaluated it experimentally on the task of solving standard MA-STRIPS benchmarks. Our results show that MA-SAM can learn the individual agents' action models from as few as a single trajectory. We show that using less than 20 trajectories, the learned action models are sufficiently powerful to solve more than 80% of the benchmark test problems in most domains. Comparison to vanilla SAM, which cannot learn from concurrent joint actions, clearly shows the benefit of our approach. Finally, we discuss methods for solving more complex settings where our previous assumptions do not hold.

Background and Problem Definition

STRIPS (1971) is a well-known language for formalizing single-agent planning problems in discrete, deterministic, and fully observable environments. Multi-agent STRIPS (MA-STRIPS) (Brafman and Domshlak 2008) is a generalization of STRIPS that formalizes multi-agent planning problems by defining a finite set of actions for each agent.

Definition 1 (MA-STRIPS). An MA-STRIPS problem is defined by a tuple $\Pi = \langle P, k, \{A_i\}_{i=1}^{k}, I, G\rangle$ where P is a set of facts (also referred as fluents), I is the initial state, G is the desired goal, k is the number of agents, and A_i is the actions agent i can perform.

The tuple $\langle P, k, \{A_i\}_{i=1}^{k}\rangle$ is called the domain. MA-STRIPS domains can be defined in a lifted manner, where facts and actions are parameterized by objects, and each object o is associated with a type, denoted type(o). A lifted fluent f is a pair $\langle \text{name}(\mathsf{f}), \text{params}(\mathsf{f})\rangle$ representing a relation over typed objects. name(f) is a symbol and params(f) is a list of types. For example, in the logistics domain from the International Planning Competition (IPC) (McDermott 2000), truck and location are types of objects, and (at?$truck$?$location$) is a lifted fluent that represents the location of a truck. A binding of a lifted fluent f is a function mapping every parameter of f to an object in O of the indicated type. A grounded fluent f is a pair $\langle \mathsf{f}, b\rangle$ where f is a lifted fluent and b is a binding for f. The term literal refers to either a fluent or its negation. The definitions of binding, lifted, and grounded fluents transfer naturally to literals. A state of the world is a set of grounded literals that includes, for every grounded fluent f, either f or $\neg f$.

A lifted action $\mathsf{a} \in \mathcal{A}$ is a pair $\langle \text{name}(\mathsf{a}), \text{params}(\mathsf{a})\rangle$ where name(a) is a symbol and params(a) is a list of types. A grounded action a is a tuple $\langle \mathsf{a}, b_{\mathsf{a}}\rangle$ where a is a lifted action and b_{a} is a binding of a. An action model for an MA-STRIPS domain is a set $M = \{M_i\}_{i=1}^{k}$ where every M_i is a pair of functions pre_{M_i} and eff_{M_i} that map agent i's actions to their preconditions and effects, respectively. Preconditions and effects are conjunctions of parameter-bound literal (pb-literal), each of which is lifted literal and a binding between the parameters of the literal and the action.

We assume in this work that for every grounded action $\langle \mathsf{a}, b_{\mathsf{a}}\rangle$, the binding b_{a} is an injective function, i.e., every parameter of a is mapped to a different object. Under this assumption, which is known as the injective binding assumption, every pair of bindings b_ℓ and b_{a} corresponds to a unique parameter binding $b_{\ell,\mathsf{a}}$. A grounded action a is applicable in a state s if every grounded literal $p \in \text{pre}_M(a)$ is satisfied by s. The result of applying a to a state s, denoted by $a(s)$, is a state composed of the facts in $\text{eff}_M(a)$ and the facts in s except those whose negations are in $\text{eff}_M(a)$. A plan π is a sequence of actions $a_1, a_2, ..., a_n$ such that (1) a_1 is applicable in the initial state I, (2) for each $1 \leq j \leq n$, a_j is applicable in the state $a_{j-1}(...a_1(I)...)$, and (3) $G \subseteq a_n(a_{n-1}(...a_1(I)...))$.

We consider the problem of solving MA-STRIPS problems without access to the agents' action model. Instead, we have access to a set of trajectories of joint actions. A trajectory $T = \langle s_0, a_1, s_1, \dots a_n, s_n\rangle$ is an alternating sequence of states $(s_0, \dots, s_n)$ and actions $(a_1, \dots, a_n)$ that starts and ends with a state. A joint action is a k-ary vector $\hat{a}$ where $\hat{a}[i]$ is either an action of agent i or NO-OP. A joint action represents the intention to execute its constituent actions concurrently, where NO-OP represents that the relevant agent does not perform any action. A joint action plan $\pi = (\hat{a}_1, \dots, \hat{a}_{|\pi|})$, is a sequence of joint actions where (1) $\hat{a}_1$ is applicable in I, (2) for each $1 \leq i \leq n$ and $1 \leq j \leq k$, $\hat{a}_i[j]$ is applicable in the state $\hat{a}_{i-1}(...\hat{a}_1(I)...)$ and (3) $G \subseteq \hat{a}_n(\hat{a}_{n-1}(...\hat{a}_1(I)...))$. A trajectory of joint actions can be created by executing a plan π that consists of joint actions to a state s. The resulting trajectory is the sequence $(s_0, \hat{a}_1, \dots, \hat{a}_{|\pi|}, s_{|\pi|})$ where $s_0 = s$ and for all $0 < i \leq |\pi|$, $s_i = \hat{a}_i(s_{i-1})$. Such a trajectory can be represented as a set of joint action triples $\left\{ \langle s_{i-1}, \hat{a}_i, s_i\rangle \right\}_{i=1}^{|\pi|}$.

Definition 2 (Model-Free Multi-Agent Planning (MF-MAP)). A MF-MAP problem is defined by a tuple $\langle D, \Pi, \mathcal{T}\rangle$, where D is an MA-STRIPS domain, Π, is an MA-STRIPS problem in D, and $\mathcal{T}$ is a set of trajectories of joint actions, created by executing plans for other problems in D.

A solution to a MF-MAP problem is a plan for the underlying MA-STRIPS problem Π. Notably, the problem-solver must find such a plan without access to the real action model of D, denoted by M^*. Note that while the given trajectories $\mathcal{T}$ are created by executing plans in the same domain, these plans differ from the MA-STRIPS problem we aim to solve (Π), starting from different initial states and achieving other goals.

Assumptions The world is deterministic and fully observable, and the given trajectories are fully observable and are created by executing sound plans for other

problems in the domain. The agents are collaborative, aiming to collaborate to achieve a shared goal, and do not explicitly aim to obfuscate their action models. The MA-STRIPS model does not support specifying over concurrently executed actions, implicitly assuming either sequential execution or that agents' actions are independent. We assume the latter case, considering the preconditions and effects of concurrently executed single-agent actions as the union of their preconditions and effects. To avoid ambiguity, we assume that concurrently executed actions are well-defined (Crosby, Jonsson, and Rovatsos 2014) in the sense that both the conjunction of their preconditions and the conjunction of their effects are each satisfiable. We discuss later possible extensions to cases where actions may not be independent and behave differently when executed concurrently (Boutilier and Brafman 2001).

Multi-Agent SAM (MA-SAM)

The approach we explore for solving MF-MAP problems is to learn an action model M' from the given set of concurrent joint action trajectories $\mathcal{T}$, and then use an off-the-shelf planner to solve the underlying MA-STRIPS problem Π using M' as the action model. To ensure that the returned plan is sound not only with respect to M' but also with respect to the real action model M^*, we require that M' is a safe action model (2021). M' is a safe action model w.r.t. an action model M if for every state s and action a, whenever a is applicable in s according to M' then (1) a is also applicable in s according to M, and (2) applying a in s results in exactly the same state according to both M and M'. We say that an action model M is safe if it is safe w.r.t. the real action model M^*. The following observation captures the relationship between a multi-agent action model's safety and its constituent single-agent action models.

Observation 1. A multi-agent action model $M' = \{M'_i\}_{i=1}^k$ is safe w.r.t $M = \{M_i\}_{i=1}^k$ iff M'_i is safe w.r.t M_i for all i.

A proof is provided in the supplementary material.

Next, we present our main contribution: the MA-SAM algorithm for learning safe action models from trajectories of joint actions. To this end, we introduce the following notation and learning rules. For a grounded action a and a grounded literal l, we denote by objects(a) and objects(l) the set of objects bound to the parameters of a and the set of objects bound to the parameters of l, respectively. We say that a grounded action a is relevant to a grounded literal l if objects(l) $\subseteq$ objects(a). For example, the grounded action (put-down a1 c) is relevant to the literal (holding a1 c). We denote by relevant$(l, \hat{a})$ the set of all actions relevant to l in the joint action $\hat{a}$.

Observation 2 (MA-SAM Rules for Lifted Actions). For any observed joint action triplet $\langle s, \hat{a}, s' \rangle$

1. If $l \notin s$ then $\langle \ell, b_{\ell,\mathsf{a}'} \rangle$ is not a precondition of a' for every $a' \in$ relevant$(l, \hat{a})$.
2. If $l \notin s'$ then $\langle \ell, b_{\ell,\mathsf{a}'} \rangle$ is not an effect of a' for every $a' \in$ relevant$(l, \hat{a})$.
3. If $l \in s' \setminus s$ then $\exists a' \in$ relevant$(l, \hat{a})$ for which $\langle \ell, b_{\ell,\mathsf{a}'} \rangle$ is an effect of a'.

These rules generalize the SAM's Inductive rules (Juba, Le, and Stern 2021) to the case of trajectories of joint actions. The MA-SAM algorithm (Algorithm 1) uses these rules to learn a safe action model, as follows.

Algorithm 1: MA-SAM Algorithm

```
for every lifted single-agent action a do
    pre(a) ← all pb-literal that can be bound to a
    eff(a) ← ∅
end for
for every lifted literal ℓ, set CNF_ℓ = ∅
for ⟨s, â, s'⟩ ∈ 𝒯 do
    Remove impossible preconditions (Rule 1)
    Add clauses to CNF_ℓ for every ℓ (Rules 2 and 3)
end for
for every lifted literal ℓ, minimize CNF_ℓ
A_safe ← every observed action
for a ∈ A_safe and ℓ ∈ L do                    ▷ a is lifted
    for every pb-literal ℓ_a not in pre(a) do
        if {IsEff(ℓ_a, a)} ∈ CNF_ℓ then
            Add ℓ_a to eff(a)
        else if {¬IsEff(ℓ_a, a)} ∉ CNF_ℓ then
            Remove a from A_safe
        end if
    end for
end for
return A_safe, pre, eff (, {CNF_ℓ}_ℓ)
```

Learning preconditions MA-SAM initially sets the preconditions of every lifted single-agent action to include every possible pb-literal. Then, for every observed action triplet $\langle s, \hat{a}, s' \rangle$ and every lifted single-agent action a such that $a \in \hat{a}$, MA-SAM removes all pb-literals that cannot be preconditions due to Rule 1 in Obs. 2. The preconditions for every lifted action a are always a (possibly not strict) superset of the preconditions for a in M^*.

Learning effects MA-SAM creates, for each lifted literal ℓ, a Conjunctive Normal Form (CNF) formula, denoted CNF$_\ell$, describing constraints on which lifted actions may have ℓ as an effect. The atoms of CNF$_\ell$ are of the form IsEff$(\ell_\mathsf{a}, \mathsf{a})$ representing that the pb-literal ℓ_a is an effect of the lifted action a.[1] These CNFs are initially empty. Then, for every observed action triplet $\langle s, \hat{a}, s' \rangle$ MA-SAM applies Rules 2 and 3 as follows. For every literal $l \notin s'$ and every action $a' \in$ relevant$(l, \hat{a})$, MA-SAM adds to CNF_ℓ the unit clause $\{\neg\text{IsEff}(\langle \ell, b_{\ell,\mathsf{a}'} \rangle, \mathsf{a}')\}$. And, for every literal $l \in s' \setminus s$ MA-SAM adds to CNF$_\ell$ the clause $\{\bigvee_{a' \in \text{relevant}(l,\hat{a})} \text{IsEff}(\langle \ell, b_{\ell,\mathsf{a}'} \rangle, \mathsf{a}')\}$. After processing all

[1] Alg. 1 treats CNF$_\ell$ as a set (conjunction) of sets (disjunctions).

the action triplets, MA-SAM minimizes the resulting CNFs by applying unit propagation. The final step in MA-SAM is to distill the effects of the agents' actions from these CNFs and identify which actions can be applied safely.

Definition 3. We say that a lifted single agent action a can be safely applied w.r.t. a set of preconditions $\text{pre}(\mathsf{a})$ and CNFs $\{\text{CNF}_\ell\}_\ell$ if for every pb-literal $\langle \ell, b_{\ell,\mathsf{a}} \rangle$, either (1) $\{\text{IsEff}(\langle \ell, b_{\ell,\mathsf{a}} \rangle, \mathsf{a})\} \in CNF_\ell$, (2) $\{\neg\text{IsEff}(\langle \ell, b_{\ell,\mathsf{a}} \rangle, \mathsf{a})\} \in CNF_\ell$, or (3) $\langle \ell, b_{\ell,\mathsf{a}} \rangle \in \text{pre}(\mathsf{a})$.

MA-SAM sets the effects of every action a that can be safely applied with respect to $\text{pre}(\mathsf{a})$ and the CNFs $\{CNF_\ell\}_\ell$ to be all the pb-literals $\langle \ell, b_{\ell,\mathsf{a}} \rangle$ for which there exists a unit clause $\{\text{IsEff}(\langle \ell, b_{\ell,\mathsf{a}} \rangle, \mathsf{a})\}$ in CNF_ℓ. Actions that cannot be safely applied are removed from the action model returned by MA-SAM, as we cannot accurately discern their effects under the learned preconditions.

Theorem 1. The action model learned using MA-SAM is safe w.r.t. the real action model (M^*), and the runtime is linear in the number of possible parameter bound literals and quadratic in the number of agents and action triplets.

See a proof sketch in the supplementary material.

Unfortunately, the sample complexity of MA-SAM, i.e., the number of trajectories it may need to learn a non-trivial action model for each agent, can be unbounded. For example, consider a pair of single-agent actions $(a_1 \quad ?x_1)$ and $(a_2 \quad ?x_2)$ that are always performed concurrently and parameterized by the same object. In this case, it is impossible to disambiguate between the effects of a_1 and a_2 and create a safe action model for each agent.

MA-SAM with Macro Actions

An alternative to MA-SAM that does not suffer from the limitation above is to use the (single-agent) SAM algorithm to directly learn an action model of joint actions instead of using MA-SAM to learn a set of single-agent action models and compose joint actions based on them. A clear limitation of this SAM over joint actions approach is that the number of joint actions is exponential in the number of agents. Yet the runtime of SAM only depends on the number of observed action triplets, not the possible number of actions. Moreover, based on the properties of SAM, we know that this approach is guaranteed to eventually learn a safe (joint) action model that, with high probability, will enable solving most problems in the domain (Juba, Le, and Stern 2021). We formalize this notion and show that, surprisingly, there are cases where this approach is, in fact, optimal.

Theorem 2. For all integers $A \geq 2$, κ, k, and $F \geq 2Ak$, and real numbers $\epsilon, \delta < 1/4$ there is a family of domains with k agents, each with A actions, and F propositional fluents such that for distributions $T(\cdot)$ supported on trajectories in which at most κ agents participate in each action (i.e., the other $k - \kappa$ take NO-OP), the sample complexity for ϵ and δ is at least $\Omega\left(\frac{1}{\epsilon}(FA^\kappa\binom{k}{\kappa} + \log\frac{1}{\delta})\right)$.

A proof is provided in the supplementary material.

SAM on joint actions is not a practical algorithm when problems contain many joint actions as it does not learn the underlying single-agent actions. Moreover, while the sample complexity of MA-SAM is worse than SAM over joint actions, our experimental results below demonstrate that MA-SAM is very effective for standard MA-STRIPS benchmarks. Next, we propose MA-SAM with Macro Actions (MA-SAM$^+$), an extension of MA-SAM that serves as a middle ground between MA-SAM and SAM over joint actions.

MA-SAM$^+$ learns single-agent actions when it can disambiguate their effects without compromising safety. When it cannot, it learns an action model for dedicated Lifted Macro Actions (LMAs) which represent the concurrent execution of a set of single-agent actions. Formally, a Lifted Macro Action (LMA) $\mathcal{A}$ is a pair $\langle A_{ca}, b_{\mathcal{A}} \rangle$ where A_{ca} is a set of lifted single-agent actions and $b_{\mathcal{A}}$ binds the parameters of these single-agent actions. In the example above, where a_1 and a_2 are always observed concurrently, MA-SAM$^+$ will learn preconditions and effects of a LMA representing the concurrent execution of a_1 and a_2. MA-SAM$^+$ augments MA-SAM by adding some LMAs to the set of actions that can be safely applied and defines appropriate preconditions and effects. Note that preconditions and effects of a LMA are pb-literals that bind parameters of a lifted literal to the parameters of the LMA's constituent lifted actions.

Algorithm 2: MA-SAM$^+$

1: $A_{safe}, \text{pre}, \text{eff}, \{\text{CNF}_\ell\}_\ell \leftarrow$ MA-SAM
2: for every relevant LMA $\mathcal{A} = \langle A_{ca}, b_{\mathcal{A}} \rangle$ do
3: $\quad \text{pre}(\mathcal{A}) \leftarrow \bigcup_{a \in A_{ca}} \text{pre}(a)$; $\text{eff}(\mathcal{A}) \leftarrow \emptyset$
4: $\quad$ for $\ell \in L$ and clause $C \in \text{CNF}_\ell$ do
5: $\quad\quad$ if C is consistent with $\mathcal{A}$ then
6: $\quad\quad\quad$ Add $\bigcup_{\text{IsEff}(\ell_\mathsf{a}, \mathsf{a}) \in C} \{\ell_\mathsf{a}\}$ to $\text{eff}(\mathcal{A})$
7: $\quad\quad$ else
8: $\quad\quad\quad$ for $\text{IsEff}(\ell_\mathsf{a}, \mathsf{a}) \in C$ s.t. $\mathsf{a} \in A_{ca}$ do
9: $\quad\quad\quad\quad$ Add ℓ_a to $\text{pre}(\mathcal{A})$
10: $\quad\quad\quad$ end for
11: $\quad\quad$ end if
12: $\quad$ end for
13: $\quad$ Add $\mathcal{A}$ to A_{safe}; $\text{pre}(\mathcal{A})$ to pre; and $\text{eff}(\mathcal{A})$ to eff.
14: end for
15: return A_{safe}, pre, eff

The pseudo-code for MA-SAM$^+$ is given in Algorithm 2. MA-SAM$^+$ starts by running MA-SAM, returning (1) a safe set of preconditions $\text{pre}(\mathsf{a})$ for every single-agent action a (line 7 in Alg. 1), (2) a CNF_ℓ for every lifted literal $\ell \in L$ (line 8), and (3) a set of single-agent actions A_{safe} that can be safely applied (lines 11-21). Then, MA-SAM$^+$ iterates over every possible

relevant LMA $\mathcal{A} = \langle A_{ca}, b_{\mathcal{A}} \rangle$ and attempts to learn preconditions and effects that allow applying it safely. $\mathcal{A}$ is called relevant in our context if there exists an action triplet $\langle s, \hat{a}, s' \rangle$ such that $\mathcal{A} \subseteq \hat{a}$ and A_{ca} includes at least one action that has not been learned safely (i.e., not in A_{safe}). MA-SAM$^+$ sets the preconditions of $\mathcal{A}$ to be the union of the preconditions of all its constituent single-agent actions according to pre (line 3). For the effects, MA-SAM$^+$ initializes eff($\mathcal{A}$) as an empty set (line 3). Then, it iterates over every lifted literal $\ell \in L$ and adds every pb-literal version of it to eff($\mathcal{A}$) if there exists a clause $C \in \text{CNF}_\ell$ that is consistent with $\mathcal{A}$ (line 6). We say that a clause C is consistent with a LMA $\mathcal{A}$ if (1) For every IsEff$(\ell_{\mathsf{a}}, \mathsf{a}) \in C$, $\mathsf{a} \in A_{ca}$, and (2) For every parameter $p \in \text{params}(\ell)$ and every pair of atoms IsEff$(\ell_{\mathsf{a}}, \mathsf{a})$ and IsEff$(\ell_{\mathsf{a}'}, \mathsf{a}')$ in C, there exists a single set in $b_{\mathcal{A}}$ that includes both $b_{\ell,\mathsf{a}}(p)$ and $b_{\ell,\mathsf{a}'}(p)$. In words, $\mathcal{A}$ is consistent with C if it includes all actions in C, and the action parameters' that are bound to the same literal's parameters will all be mapped to the same object when $\mathcal{A}$ is applied. This means applying $\mathcal{A}$ must have the effect specified in C, and we can safely add it to eff($\mathcal{A}$). If $\mathcal{A}$ is not consistent with any $C \in \text{CNF}_\ell$, it does not guarantee that $\mathcal{A}$ does not achieve ℓ. Thus, to preserve safety, MA-SAM$^+$ adds to the preconditions of $\mathcal{A}$ every pb-literal $\ell_{\mathcal{A}}$ that MA-SAM$^+$ cannot determine if it is an effect of $\mathcal{A}$ or not (line 9).

The worst-case runtime of MA-SAM$^+$ is exponential in the number of agents since it searches for all possible LMAs in a brute-force manner. It may be possible to restrict the number of required LMAs by analyzing $\{\text{CNF}_\ell\}_\ell$. In addition, MA-SAM$^+$ requires a dedicated planner that can choose to perform the learned LMAs. Thus, we only implemented MA-SAM in our experimental results below, and leave efficient implementations of MA-SAM$^+$ and the design of an appropriate planner for future work. [2]

Experimental Results

We conducted experiments on domains from the publicly available Competition of Distributed and Multiagent Planners (CoDMAP) benchmark of MA-STRIPS problems (Štolba, Komenda, and Kovacs 2016). This benchmark includes 12 MA-STRIPS domains with 20 problems each. We used seven domains from this benchmark, namely Blocksworld, Depots, Driverlog, Logistics, Rovers, Satellite, and Sokoban. The other domains either fail to satisfy the injective binding assumption, or our planners could not solve enough problems even with the real action model.

In each experiment, we sampled a set of trajectories in a domain, ran the evaluated learning algorithm with these trajectories, and then attempted to solve the remaining problems in that domain using the learned action model. We followed a 5-fold cross-validation methodology by repeating each experiment 5 times, sampling different trajectories for learning and testing. All experiments ran on a Linux machine with 8 cores and 16 GB of RAM.

Generating trajectories of joint actions To generate trajectories for learning, we generated plans for the problems in the evaluated domain using two MA-STRIPS solvers: GPPP (2017) with landmarks and FF heuristics and MAFS (2014) with the FF heuristic. Like most existing MA-STRIPS solvers, these MA-STRIPS solvers output a plan π which is a sequence of single-agent actions. To create trajectories with non-trivial joint actions, we grouped maximal consecutive sequences of single-agent actions that form well-defined joint actions (Crosby, Jonsson, and Rovatsos 2014).

In some domains, the resulting trajectories of joint actions contained fewer than 10% action triplets with non-trivial joint actions, i.e., joint actions that include more than one action that is not a NO-OP. Since our contribution is learning from concurrently executed actions, we conducted a second set of experiments in which we modified the benchmark domain to encourage non-trivial joint actions. Specifically, we added the predicate dummy-additional-predicate with no parameters, and the agents' actions do not add or delete it. Then, we added two new actions to each agent dummy-add-predicate-action and dummy-del-predicate-action that have no preconditions, and they add or delete the newly added predicate, respectively. To create trajectories containing these new actions, we randomly added the new actions with the original ones to compose joint actions while the resulting plans remained well-defined. We set the probability of adding one of the new actions to 0.65 for each joint action in the plan sequence. This means, on average, at least 65% of the transitions in our trajectories include a non-trivial joint action. We chose such a high percentage of non-trivial joint actions to challenge the algorithms and observe the necessity of supporting concurrency in multi-agent planning. Finally, we randomly selected problems and added the newly defined predicate to the goal state. This addition prevents a planner from solving problems unless the action model it uses includes actions that affect dummy-additional-predicate, i.e., by learning from non-trivial joint actions.

Table 1 presents characteristics of the domains used in our experiments. Columns $|A|$, $|P|$, $arity(p)$, $arity(a)$, "max $|A_i|$", and $|\mathcal{T}|$ contain the number of lifted actions (not including those newly added), the number of lifted fluents, the maximum arity of fluents, the maximum arity of actions, the maximum number of agents acting in a problem, and the total number of trajectories available respectively. Columns "$|t|$", "%SA$_1$", and "%SA$_2$" present the total number of action triplets in the training trajectories and the percentage of triplets that consist only of trivial joint actions.

Evaluation Metrics We measured the number of problems solved with the learned action model and the precision and recall of the learned action model. To

[2] We will publish MA-SAM's code once the paper is accepted.

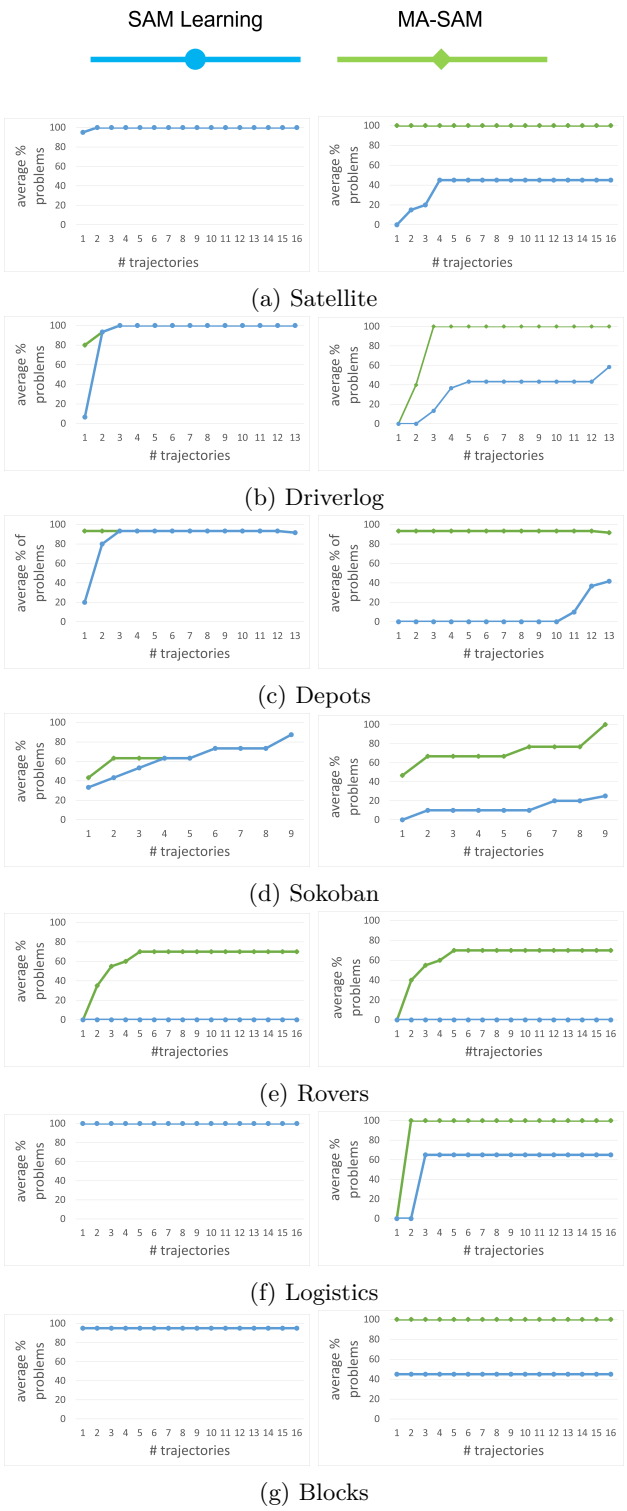

Figure 1: % of the test set problems solved by MA-SAM (green diamond) and SAM (blue circle) in each domain. (left) results without the additional actions to encourage concurrency. (right) results with encouraged concurrency.

Domain	$\|A\|$	$\|P\|$	max $arity(p)$	max $arity(a)$	max $\|A_i\|$	$\|\mathcal{T}\|$	$\|t\|$	$\%SA_1$	$\%SA_2$
blocksworld	4	5	2	3	4	20	1314	91	34
depots	5	7	3	4	12	16	506	67	23
driverlog	6	6	2	4	8	16	277	94	28
logistics	5	6	3	4	7	20	686	74	26
rovers	9	25	2	6	10	20	833	88	31
satellite	5	8	2	4	10	20	1,053	99	35
sokoban	3	6	3	6	4	11	406	78	29

Table 1: Characteristics of the domains in our experiments.

measure the number of problems solved, we ran Fast-Downward (Helmert 2006) with the learned action model and counted the number of problems solved within a time limit of 60 seconds. In our experiments, we observed that the effects are learned perfectly and very quickly, and , we focus on preconditions' precision and recall. We distinguish between the syntactic precision and recall of the preconditions and their semantic precision and recall, denoted by P^{syn}_{pre}, R^{syn}_{pre}, P^{sem}_{pre} and R^{sem}_{pre} respectively. The syntactic precision and recall values measure the textual difference between the real and learned domains. These metrics may be misleading as two action models may be different syntactically yet still allow solving the same number of problems. The semantic precision and recall of the preconditions address this limitation and is computed as follows:

$$P^{sem}_{\text{pre}}(a) = \frac{|app_{M^*}(a) \cap app_M(a)|}{|app_M(a)|}$$

$$R^{sem}_{\text{pre}}(a) = \frac{|app_{M^*}(a) \cap app_M(a)|}{|app_{M^*}(a)|}$$

where $app_M(a)$ denotes the states in a set of trajectories where a is applicable according to the action model M. Since MA-SAM learns a safe action model, the syntactic recall and the semantic precision are always 1, so we only present results for syntactic precision and semantic recall. There is no natural baseline to compare with MA-SAM since all other works on learning action models do not support learning from trajectories with joint actions. Thus, as a baseline, we compared MA-SAM with the single-agent SAM algorithm, which can only learn from action triplets in which the joint action contains exactly one action.

Results Figure 1 presents the average percentage of problems solved as the number of input trajectories varies for each domain in both settings. In the first experiment, in all domains except Rovers, both MA-SAM and SAM were eventually able to solve over 85% of the test problems. In Blocksworld and Logistics, both algorithms learned a correct action model after one trajectory and solved almost all test problems. The results on Driverlog, Depots, Sokoban, and Rovers highlight that MA-SAM learns faster than SAM, as expected. The weaker performance of both algorithms on Rovers is due to the size of the domain: it has significantly more lifted fluents and actions than all other domains. The advantage of MA-SAM over SAM is highlighted by the

results on the second set of experiments, shown in Figure 1 (right). For example, in Blocksworld, SAM could not solve more than 45% of the test problems while MA-SAM had 100% success.

Domain	P^{syn}_{pre}	R^{sem}_{pre}
Blocksworld	0.60	1.00
Depots	0.68	1.00
Driverlog	0.59	1.00
Logistics	0.66	1.00
Rovers	0.62	0.88
Satellite	0.70	0.99
Sokoban	0.68	0.99

Table 2: Average syntactic precision and semantic recall of actions' preconditions.

Table 2 shows the average syntactic precision and semantic recall for preconditions (P^{syn}_{pre} and R^{sem}_{pre}) for the action model learned by MA-SAM for every domain given all the available trajectories. We show results only for the second set of experiments since it presented more challenges to both algorithms. Interestingly, the syntacicantly different. This difference highlights that while MA-SAM's conservative approach to learning preconditions indeed learns more preconditions than those appearing in the PDDL domain, in most cases, these additional preconditions do not affect the Here too, Rovers pose the greatest challenge to MA-SAM

MA-SAM and MA-SAM$^+$ are designed for cases where single-agent actions are independent. In some domains, differently when executed concurrently (2001). This section discusses how MA-SAM and MA-SAM$^+$ can be extended to support such domains. Such domains may include conflicting actions, i.e., where the preconditions over rently, as well as collaborative actions, i.e., sets of acstituent single-agent actions. The SAM over joint ac for such domains. This algorithm cannot scale beyond be actions can allow more efficient learning.

Conflicting Actions Consider domains with conflicting but not collaborative actions. In such domains, MA-SAM and MA-SAM$^+$ can be used to learn actions' effects, but learning preconditions is challenging. Without any Thus, way to learn a safe action model.

More efficient learning can be achieved by restricting conflicts are defined for at most n concurrently executed actions (e.g., only pair-wise conflicts). In that size n and can then generalize to joint actions containing more single-agent actions. For example, given three actions a_1, a_2, a_3 and $n = 2$ if we observe the joint actions: $\langle a_1, a_2 \rangle, \langle a_1, a_3 \rangle, \langle a_2, a_3 \rangle$, we can infer that the joint execution of $\langle a_1, a_2, a_3 \rangle$ is safe. Assuming we have $k \gg n$ agents, this approach significantly reduces the ing the safety property.

conflict caused by violating the concurrency constraint (Boutilier and Brafman 2001). Recall that concurrency constraints disallow some joint actions, either because two or more actions cannot be performed in parallel or, on the contrary, because some actions must be executed in parallel (Boutilier and Brafman 2001; Crosby, Jonsson, and Rovatsos 2014). Under such a setting, we can learn an approximation to the lower and upper bounds for the concurrency constraint of the actions' objects. This can be achieved by assigning a lower and upper bound on the actions' parameters based on the observations in which the actions are being applied. I.e., for each joint action, we can calculate the number Corollary 2 [3] also addresses actions with concurrency constraints.

-defined joint actions (Boutilier and Brafman 2001). These are joint actions in which some of the actions' preconditions or effects may conflict. We propose a conflict inference stage for MA-SAM to address this setting. learned actions during the learning stage, using the observation objects, and inferring whether there are two grounded actions with contradicting preconditions or effects. If so, we can deduce that these two actions can same objects and should always be independent. We can

Collaborative Actions Finally, we address learning collaborative actions. As before, safely learning all posserved collaborative effects cannot be inferred, to safely observe all possible joint executions of actions. On the as those presented in (Shekhar and Brafman 2020) is already supported in MA-SAM$^+$ since such actions are just additional macro actions with the agents as part of

create collaborative effects, n, we observe that collaborative effects are conditional effects of the joint actions. concurrently on the same objects, i.e., when the additional condition stating the single-agent actions com pear fects, we use MA-SAM$^+$ to learn all possible joint actions up to size n while adding the functionality presented in (Mordoch et al. 2023). The result of applying MA-SAM$^+$ while learning the conditional effects will be new macro actions that contain conditional effects similar to those presented in (Hofmann, Niemueller, and Lakemeyer 2020).

This work explored the problem of learning action models in a multi-agent setting. Unlike previous works, we learned the single agents' action models from trajectories of joint actions. We presented MA-SAM, an algorithm capable of learning from such trajectories. We observed that in some cases, MA-SAM encounters amties, we presented SAM over joint actions, which cannot scale but has a bounded sample complexity. Then, we presented MA-SAM$^+$, which extends MA-SAM by learning lifted macro actions (LMAs) in cases where it cannot safely identify single-agent actions' effects. All algorithms return a safe action model, which means To evaluate MA-SAM, we experimented with seven classical multi-agent planning benchmarks. We showed that given a small number of trajectories, MA-SAM creates

[3] Presented in the supplementary material.

problems. We also discussed different approaches act. Future work will explore such settings as well as efficient implementations of MA-SAM$^+$.

References

Allouche, M. K.; and Boukhtouta, A. 2010. Multi-agent coordination by temporal plan fusion: Application to combat search and rescue. Information Fusion, 11(3): 220–232.

Azadeh, K.; De Koster, R.; and Roy, D. 2019. Robotized and automated warehouse systems: Review and recent developments. Transportation Science, 53(4): 917–945.

Barrett, S.; and Stone, P. 2012. An analysis framework for ad hoc teamwork tasks. In International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 357–364.

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov decision processes. Mathematics of operations research, 27(4): 819–840.

Boutilier, C.; and Brafman, R. I. 2001. Partial-order planning with concurrent interacting actions. Journal of Artificial Intelligence Research, 14: 105–136.

Brafman, R. I.; and Domshlak, C. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In ICAPS, 28–35.

Brafman, R. I.; and Domshlak, C. 2013. On the complexity of planning for agent teams and its implications for single agent planning. Artificial Intelligence, 198: 52–71.

Crosby, M.; Jonsson, A.; and Rovatsos, M. 2014. A Single-Agent Approach to Multiagent Planning. In ECAI, 237–242.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial intelligence, 2(3-4): 189–208.

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

Hofmann, T.; Niemueller, T.; and Lakemeyer, G. 2020. Macro operator synthesis for adl domains. In ECAI 2020, 761–768. IOS Press.

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.

Juba, B.; and Stern, R. 2022. Learning Probably Approximately Complete and Safe Action Models for Stochastic Worlds. In AAAI.

Knepper, R. A.; Layton, T.; Romanishin, J.; and Rus, D. 2013. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In 2013 IEEE International conference on robotics and automation, 855–862. IEEE.

Lesser, V.; Ortiz Jr, C. L.; and Tambe, M. 2003. Distributed sensor networks: A multiagent perspective, volume 9. Springer Science & Business Media.

Maliah, S.; Shani, G.; and Stern, R. 2016. Collaborative privacy preserving multi-agent planning. Autonomous Agents and Multi-Agent Systems, 1–38.

Maliah, S.; Shani, G.; and Stern, R. 2017. Collaborative privacy preserving multi-agent planning. Autonomous Agents and Multi-Agent Systems, 31(3): 493–530.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. AI Magazine, 21(2): 13.

Mordoch, A.; Juba, B.; and Stern, R. 2023. Learning Safe Numeric Action Models. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 37, 12079–12086.

Mordoch, A.; Stern, R.; Scala, E.; and Juba, B. 2023. Safe Learning of PDDL Domains with Conditional Effects.

Nissim, R.; and Brafman, R. 2014. Distributed heuristic forward search for multi-agent planning. Journal of Artificial Intelligence Research, 51: 293–332.

Shekhar, S.; and Brafman, R. I. 2020. Representing and planning with interacting actions and privacy. Artificial Intelligence, 278: 103200.

Stern, R.; and Juba, B. 2017. Efficient, Safe, and Probably Approximately Complete Learning of Action Models. In the International Joint Conference on Artificial Intelligence (IJCAI), 4405–4411.

Štolba, M.; Komenda, A.; and Kovacs, D. 2016. Competition of distributed and multiagent planners (codmap). In AAAI Conference on Artificial Intelligence, volume 30.

Tambe, M. 1997. Towards flexible teamwork. Journal of artificial intelligence research, 7: 83–124.

Verma, P.; Marpally, S. R.; and Srivastava, S. 2021. Asking the right questions: Interpretable action model learning using query-answering. In AAAI.