# Team-Imitate-Synchronize for Solving Dec-POMDPs *

Eliran Abdoo, Ronen I. Brafman✉, Guy Shani and Nitsan Soffair

Ben-Gurion University of the Negev
{eliranab,brafman,shanigu,soffair}@bgu.ac.il

**Abstract.** Multi-agent collaboration under partial observability is a difficult task. Multi-agent reinforcement learning (MARL) algorithms that do not leverage a model of the environment struggle with tasks that require sequences of collaborative actions, while Dec-POMDP algorithms that use such models to compute near-optimal policies, scale poorly. In this paper, we suggest the Team-Imitate-Synchronize (TIS) approach, a heuristic, model-based method for solving such problems. Our approach begins by solving the joint team problem, assuming that observations are shared. Then, for each agent we solve a single agent problem designed to imitate its behavior within the team plan. Finally, we adjust the single agent policies for better synchronization. Our experiments demonstrate that our method provides comparable solutions to Dec-POMDP solvers over small problems, while scaling to much larger problems, and provides collaborative plans that MARL algorithms are unable to identify.

## 1 Introduction

Problems that require collaborative effort by several agents, operating under partial observability, are extremely challenging. Such problems can be tackled by a centralized planning algorithm that creates a policy for each agent. Then, each agent executes its policy in a distributed manner, restricting communication with other agents to explicit actions dictated by the policy.

Recently, cooperative multi agent problems are often tackled by deep multi-agent reinforcement learning (MARL), often under the term *centralized learning for decentralized execution*, showing impressive improvements [16, 18]. RL is able to learn a policy directly, without requiring access to a model of the environment's dynamics. In the single-agent case, model-free RL methods are sometimes employed even when a model exists, because in many problems, a specification of a policy can be orders of magnitude smaller than a model of the environment.

However, in many MA domains, a sequence of collaborations, conditioned on appropriate observations, is needed to complete a task and earn a reward. MARL algorithms must explore the policy space, blindly at first, to identify such beneficial behaviors. As we show in this paper, current MARL algorithms have significant difficulty discovering such sequences by pure exploration.

* Supported by ISF Grants 1651/19 and 1210/18, Ministry of Science and Technology's Grant #3-15626 and the Lynn and William Frankel Center for Computer Science.

On the other side of the spectrum lie algorithms that rely on a complete specification of the environment, typically as a Dec-POMDP model [3]. Given such a specification the agents can identify good behaviors more easily. Some Dec-POMDP solvers can compute solutions with optimality guarantees or error bounds. But these solvers have difficulty scaling up beyond very small problems – not surprising given the NEXP-Time hardness of this problem [3].

In this paper we suggest a new approach for solving MA problems *given* a model. Like Deep MARL methods, our approach does not provide optimality guarantees, yet scales significantly better than existing Dec-POMDP algorithms. Unlike MARL methods, we can use the world model to better guide the agents towards complex beneficial behaviors. This allows us to solve problems that require a sequence of collaborative actions, where MARL methods utterly fail.

Our approach, Team-Imitate-Synchronize (TIS) works in 3 phases. First, we solve a *team POMDP* in which every agent's observations are implicitly available to the other agents. Hence, all agents share the same belief state. The solution to the team POMDP is typically not executable because it may condition an agent's actions on observations made by other agents. Hence, in the next step, TIS tries to produce a policy for each agent that *imitates* that agent's behavior within the team policy. The resulting policies are executable by the agents, as they depend on their own observations only. However, they are not well synchronized. The last step improves the synchronization of the timing of action execution by different agents, while still relying on information available to that agent only. In the Dec-POMDPs with a *no-op* action we consider here, this can be done by delaying the execution of particular parts of the policy.

TIS is a general approach for solving Dec-POMDPs — there are different ways of instantiating the Imitate and Synchronize steps, and we offer here a simple, heuristic instantiation. We create a specific imitation-POMDP for each agent in which it receives reward for behaving similarly to its behavior in the team policy. The synchronization step employs a heuristic approach that analyzes the agents' policy trees to improve synchronization. That being said, our chosen methods for these steps enable us to handle many MA problems that cannot be currently solved by any other method. TIS does not provide optimality guarantees because the Team step solves a relaxed version of the original multi-agent problem. We know that an optimal solution to a relaxed problem may not be refinable to an optimal solution of the original problem (e.g., see the case of hierarchical planning and RL [7]). Yet, relaxation-based methods offer a very practical approach in many areas.

We experiment on 3 problems, comparing our approach to exact and approximate Dec-POMDP solution methods, as well as MARL algorithms. We demonstrate that TIS scales significantly better than all other methods, especially in domains that require non-trivial coordination of actions. Such collaborations include both the ability to order actions properly, so that one agent's actions help set up conditions needed for the success of other agents' actions, and the ability to perform appropriate actions concurrently. Code and domain encodings are available at https://github.com/neuronymous/decpomdp-solver.

## 2 Background

*POMDPs:* A POMDP models single-agent sequential decision making under uncertainty and partial observability. It is a tuple $P = \langle S, A, T, R, \Omega, O, \gamma, h, b_0 \rangle$. $S$ is the set of states. $A$ is the set of actions. $T(s, a, s')$ is the probability of transitioning to $s'$ when applying $a$ in $s$. $R(s, a)$ is the immediate reward for action $a$ in state $s$. $\Omega$ is the set of observations. $O(a, s', o)$ is the probability of observing $o \in \Omega$ when performing $a$ and *reaching* $s'$. $\gamma \in (0, 1)$ is the discount factor. $h$ is the planning horizon. A *belief state* is a distribution over $S$, with $b_0 \in \prod(S)$ denoting the initial belief state.

We focus on factored models where each state is an assignment to some set of variables $X_1, \ldots, X_k$, and each observation $\Omega$ is an assignment to observation variables $W_1, \ldots, W_d$. Thus, $S = Dom(X_1) \times \cdots \times Dom(X_k)$ and $\Omega = Dom(W_1) \times \cdots \times Dom(W_d)$. In that case, $\tau$, $O$, and $R$ can be represented compactly by, e.g., a dynamic Bayesian network [4].

For ease of representation, we assume that actions are either sensing actions or non-sensing actions. Sensing actions do not modify the state of the world, and may result in different observations in different states. An agent that applies a non-sensing action always receives the *null-obs* observation. We assume that every action has one effect on the world that we consider as its successful outcome, while all other effects are considered failures. Both assumptions are realistic in many domains. Both can be removed, at the cost of a more complex algorithm.

A solution to a POMDP is a *policy* that assigns an action to every history of actions and observations (*AO-history*). It is often represented using a *policy tree* or *graph* (a.k.a. finite-state controller). Each vertex is associated with an action, and each edge is associated with an observation.

A *trace* $T$ is a sequence of quintuplets $e_i = (s_i, a_i, s'_i, o_i, r_i)$, where $s_i$ is the state in step $i$, $a_i$ is the action taken in step $i$; $s'_i = s_{i+1}$ is the resulting state, and $o_i$ and $r_i$ are the observation and reward received after taking action $a_i$ in $s_i$ and reaching $s'_i$. For brevity, in our description, we typically ignore $s'_i$ and $r_i$.

*Dec-POMDPs:* Dec-POMDPs model problems where $n > 1$ fully cooperative agents seek to maximize the expected sum of rewards received by the team. The agents act in a distributed manner and obtain different observations, so their information states may differ. Formally, a Dec-POMDP for $n$ agents is a tuple $P = \langle S, A = \times_{i=1}^{n}\{A_i\}, T, R, \Omega = \times i = 1^n\{\Omega_i\}, O, \gamma, h, b_0 \rangle$. The components are similar to those of a POMDP with the following differences: each agent $i$ has its own set of actions $A_i$ and its own set of observations $\Omega_i$. These sets define the *joint-action* set $A = A_1 \times A_2 \times .. \times A_n$ and the *joint-observation* set $\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_n$. All other elements are defined identically as in a POMDP w.r.t. the set of joint-actions and joint-observations. We assume $A_i$ *always* contains a *no-op* action that does not modify the state of the world nor generates any meaningful observation. (This essentially implies that there are no exogenous processes.) We also use the *no-op*s for *reward-calibration*: a joint action consisting of *no-op*s only has a reward of 0. The agents share the initial belief state, $b_0$. However, during execution, agent $i$ receives only its component

$\omega_i$ of the joint observation $\omega = (\omega_1, \ldots, \omega_n)$. A solution to a Dec-POMDP is a *set* of policies $\rho_i$ (as defined for POMDP), one for each agent.

Dec-POMDPs can use a factored specification [15], although most work to date uses the flat-state representation. An important element of a factored specification is a compact formalism for specifying joint-actions. If each agent has $k$ actions, then, in principle, there are $O(k^n)$ possible joint actions. Yet, in many problems of interest most actions do not interact with each other. If $a \in A_i$ is an action of agent $i$, we identify $a$ with the joint action (no-op, $\ldots, a, \ldots,$ no-op). Actions $a_i \in A_i$, $a_j \in A_j$ are said to be non interacting, if their effect distribution, when applied jointly (i.e., (no-op, $\ldots, a_i, \ldots, a_j \ldots,$ no-op)), is identical to their effect distribution when applied sequentially. Thus, our specification language focuses on specifying the effects of single-agent actions and specific combinations of single-agent actions that interact with each other, which we call *collaborative* actions [2]. As above, we identify the collaborative action of a group of agents with the joint action in which all other agents perform a *no-op*. Then, we can decompose every joint action into some combination of non-interacting single-agent and collaborative actions, defining its dynamics.

*Example 1.* Our running example consists of a 2-cell box-pushing domain, with cells $L$(left) and $R$(right), two agents, and two boxes. $B_1$ is light, and $B_2$ is heavy (Figure 1). The state is composed of 4 state variables: the location of each box – $(X_{B1}, X_{B2})$ – and the location of each agent – $(X_{A1}, X_{A2})$. In addition, there are two observation variables for each agent $(\omega_1^i, \omega_2^i)$. $\omega_j^i$, indicates to $Agent_i$ whether it is co-located with $B_j$. Initially, $A_1$ and $B_1$ are at $L$ and $A_2$ and $B_2$ are at $R$. The goal is to swap the boxes, i.e. $(X_{B1} = R, X_{B2} = L)$. Agents can move, push a box, or sense their current cell for a box. Move and Push can be done in any direction. Push actions fail with some probability, and a single-agent cannot succeed pushing the heavy box. The action in which both agents push a heavy box is modeled as a *collaborative-push* action.

*Public, Private and Relevant Variables:* A state variable $X_i$ is *affected* by $a$ if there is some state $s$ for which there is a non zero probability that the value of $X_i$ changes following $a$. We denote the variables affected by $a$ by $\mathit{eff}(a)$. $X_i$ is *affected* by agent $j$, if $X_i \in \mathit{eff}(a)$ for some $a \in A_j$. $X_i$ is called *public* if it is affected by two or more agents, that is, there exist $j \neq k$ and actions $a \in A_j$, $a' \in A_k$ such that $X_i \in \mathit{eff}(a) \cap \mathit{eff}(a')$. An action $a$ is called *public* if one of its effects is public. Otherwise, $a$ is *private*. Thus, collaborative actions are always public. Sensing actions are private, by nature. Here, we also assume they are non-collaborative, i.e., they affect one agent's observation variables only.

A state variable $X_i$ is an *influencer* of $a$ if $a$ behaves differently given different values of $X_i$. That is, if there are two states $s_1, s_2$ that differ only in the value of $X_i$ such that $R(s_1, a, s') \neq R(s_2, a, s')$, or $T(s_1, a, s') \neq T(s_2, a, s')$ for some state $s'$, or in the case of sensing actions, $O(a, s_1, o) \neq O(a, s_2, o)$ for some observation $o$ . We denote influencers of $a$ by $\mathit{inf}(a)$. We refer to the union of the influencers and effects of $a$ as the *relevant* variables of $a$, denoted $\mathit{rel}(a)$.

*Example 2.* In our running example, $X_{B1}$ and $X_{B2}$ are the public variables, as they are the effects of both agents' push actions. $X_{A1}$, $X_{A2}$ are private variables of agent 1 and agent 2, respectively, as they are the effect of a single agent's move action. $\omega_j^i$ is private to $Agent_i$, being the effect of its sensing actions. Actions move and sense are private, while the push actions are public.

## 3 TIS – Team-Imitate-Synchronize

We begin with a high level description of the major components of TIS. Then, we explain each step in more depth.

**1. Team Problem:** Given a Dec-POMDP model $P$ as input, this step outputs a near-optimal policy $\pi_{team}$ for the *team* POMDP $P_{team}$. $P_{team}$ is identical to $P$ but ignores the underlying multi-agent structure. That is, actions and observations in $P_{team}$ are the joint actions and the joint observations of $P$ with the same transition and reward function. $P_{team}$ models a single agent that controls all agents and receives the joint observations. We obtain $\pi_{team}$ by solving $P_{team}$ using a POMDP solver. This can be a model-based solver or an RL algorithm that can handle POMDPs, such as DRQN [8].

**2. Generate Tree or Traces:** Some POMDP solvers output a policy tree. If this tree is very large, we approximate the set of path in it by simulating $\pi_{team}$ on $P_{team}$, obtaining a set T of execution traces.[1]

**3. Imitate:** Given the Dec-POMDP model $P$ and the traces T as input, in this step every agent tries to imitate its behavior in the team policy. This is a non-standard imitation learning problem. First, each agent has access to less information than the expert (=team). Second, the agent can sometimes obtain additional information by applying sensing actions that are not part of the team policy. Third, how good one agent's imitation policy depends on how other agents imitate their part of the team policy. While this can be a very interesting imitation learning problem, instead, in this paper we use our model to construct an imitation POMDP $P_i$, for each agent $i$. $P_i$'s dynamics are similar to $P_{team}$, ignoring certain variables and actions that are not relevant for agent $i$, and the observations available to other agents. $P_i$ rewards the agent when its action choice is similar to that which appears in a comparable trace of $\pi_{team}$. Its solution, $\pi_i'$, is the basis for $i$'s policy.

**4. Synchronize:** Given the agents' policies, $\{\pi_i'\}_{i=1}^n$, generate a policy graph for each agent and compute modified single-agent policies $\{\pi_i\}_{i=1}^n$ aiming at (probabilistically) better coordination between agents. This is done by inserting additional *no-op* actions to agents' policies to affect the timing of action execution. Notice that in Dec-POMDPs, one can always insert a *no-op* action. Whether this helps or not depends on what other agents do at the same time. Specifically, we focus on improving the probability that individual parts of a collaborative

[1] Our implementation uses the simulation function of the SARSOP solver. We precompute sample size based on concentration bounds that ensure that distribution over initial state will match the true belief state.

Table 1: Two example traces. $P, S, M, CoP$ abbreviate *Push, Sense, Move, Collaborative-Push*

| | $X_{A1}$ | $X_{A2}$ | $X_{B1}$ | $X_{B2}$ | $a_1$ | $a_2$ | $\omega_1^1$ | $\omega_2^1$ | $\omega_1^2$ | $\omega_2^2$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1.1 | L | L | R | R | $PushRight(A_1, B_1)$ | *no-op* | $\phi$ | $\phi$ | $\phi$ | $\phi$ |
| 1.2 | L | R | R | R | $SenseBox(A_1, B_1)$ | *no-op* | no | $\phi$ | $\phi$ | $\phi$ |
| 1.3 | L | R | R | R | $MoveRight(A_1)$ | *no-op* | $\phi$ | $\phi$ | $\phi$ | $\phi$ |
| 1.4 | R | R | R | R | $CPushLeft(A_1, B_2)$ | $CPushLeft(A_2, B_2)$ | $\phi$ | $\phi$ | $\phi$ | $\phi$ |
| 1.5 | R | R | R | L | *no-op* | $SenseBox(A_2, B_2)$ | $\phi$ | $\phi$ | $\phi$ | no |
| 2.1 | L | L | R | R | $PushRight(A_1, B_1)$ | *no-op* | $\phi$ | $\phi$ | $\phi$ | $\phi$ |
| 2.2 | L | R | R | R | $SenseBox(A_1, B_1)$ | *no-op* | no | $\phi$ | $\phi$ | $\phi$ |
| 2.3 | L | R | R | R | $MoveRight(A_1)$ | *no-op* | $\phi$ | $\phi$ | $\phi$ | $\phi$ |
| 2.4 | R | R | R | R | $CPushLeft(A_1, B_2)$ | $CPushLeft(A_2, B_2)$ | $\phi$ | $\phi$ | $\phi$ | $\phi$ |
| 2.5 | R | R | R | R | *no-op* | $SenseBox(A_2, B_2)$ | $\phi$ | $\phi$ | $\phi$ | yes |
| 2.6 | R | R | R | R | $CPushLeft(A_1, B_2)$ | $CPushLeft(A_2, B_2)$ | $\phi$ | $\phi$ | $\phi$ | $\phi$ |
| 2.7 | R | R | R | L | *no-op* | $SenseBox(A_2, B_2)$ | $\phi$ | $\phi$ | $\phi$ | no |

action will be synchronized and that the action order in $\pi_{team}$ between agents is maintained. $\{\pi_i\}_{i=1}^n$ is the final output of the entire algorithm.

Steps 1 and 2 are straightforward. Below, we detail Steps 3 and 4.

*Example 3.* A possible team policy for our example is shown in Figure 1a. Edges are labeled by the joint observations of the team. $A_1$ begins by pushing $B_1$ to the right, then senses whether it succeeded. It then moves right to assist $A_2$ to push the heavy box, $B_2$, to the left. $A_2$ then senses for success. As observations are shared in $P_{team}$, $A_1$ is also aware of the resulting observation. Two example traces are shown in Table 1.

### 3.1 Generating the Individual Agent POMDPs

We now aim to generate agent policies that, combined, will behave similarly to the team policy. This can be achieved in several ways. For example, we could try to imitate the behavior based on the agents' individual belief state. Here, we suggest a heuristic approach, motivated by a simple intuition. We design for each agent $i$ a POMDP $P_i$, in which the world dynamics remains the same, but agent $i$ is rewarded whenever it imitates its role in the team policy. That is, $i$ is rewarded when executing an action similarly to the public plan.

We focus on imitating public actions as they influence other agents. We wish to reward an agent when it executes a public action in a context in which it was applied in the collected traces T. Hence, we define a context $c$ for an action $a$, and reward $i$ only when it applies $a$ in $c$. Public actions not encountered in T are not relevant for imitation, and thus we remove them from the imitation POMDPs, $P_i$. For private actions, we maintain the same reward as in $P$.

Defining the *context* in which a public action $a$ is applied in a trace to be the state it was applied in, is too narrow. We must generalize the context to capture

only the relevant variables within the state. To better generalize from the states at which $a$ was executed in T, we remove irrelevant variables from the context.

**Definition 1.** *The* context *$c$ of action $a$ of agent $i$ in state $s$ is the projection of $s$ to the set of variables consisting of all public variables and any private variable of $i$* relevant *to $a$. The pair $\langle c, a\rangle$ is called a* contexted action *(CA).*

$\mathrm{CA}_i$*, the set of contexted actions for agent $i$, is the union of all contexted actions $\langle c, a\rangle$ for all state,action pairs for agent $i$ appearing in any trace in* T.

*Example 4.* The public actions of $A_1$ in the trace elements shown in Table 1 are $PushRight(A_1, B_1)$ in 1.1,2.1, and $CPushLeft(A_1, B_2)$ in 1.4,2.4,2.6. These actions appear multiple times in identical contexts. Context of $PushRight(A_1, B_1)$ contains the public variable $X_{B1}$, $X_{B2}$, and $X_{A1}$ which is the only private relevant variable of $PushRight(A_1, B_1)$. The context of $CPushLeft(A_1, B_2)$ for $A_1$ is identical. Thus: $CA_1 = \{\langle\langle X_{A1} = L, X_{B1} = L, X_{B2} = R\rangle, PushRight(A_1, B_1)\rangle$, $\langle\langle X_{A1} = R, X_{B1} = R, X_{B2} = R\rangle, CPushLeft(A_1, B_2)\rangle\}$. Also, $CA_2$=$\{\langle\langle X_{A2} = R, X_{B1} = R, X_{B2} = R, \rangle, CPushLeft(A_2, B_2)\rangle\}$.

Encouraging the execution of a public action in an appropriate context is insufficient. We must also discourage execution of public actions outside their context. Public actions modify public variables' values, which may cause future actions by other agents to have undesirable outcomes that differ from the team plan. Hence, we associate negative reward with out-of-context public actions.

This must be done carefully. $P_i$ contains the non-sensing actions of all agents. This helps the synchronizing agent $i$'s policy with those of other agents. That is, the agent has to simulate in its policy actions of other agents that are needed for its own reward. Thus, it must time its actions appropriately w.r.t. other agents' actions (simulated by it), which leads to better coordination. However, $P_i$ does not contain the sensing actions of other agents. Therefore, we should not penalize it for performing actions when the value of a variable it cannot gain information on is "wrong". For this reason, we define a relaxed context.

**Definition 2.** *Let $X^i_{obs}$ be the set of variables that agent $i$ can learn about through, possibly noisy, observations. The* relaxed context *$c'$ of a contexted action $\langle c, a\rangle$ is the projection of $c$ to the set of public variables and $X^i_{obs}$.*

That is, $c'$ is obtained from $c$ by ignoring some context variables. Therefore, $\neg c' \rightarrow \neg c$, and fewer states are associated with this penalty than had we applied it to any state *not* satisfying $c$. This leads to the following definition of the factored single-agent POMDP $P_i$, solved by agent $i$.

**Actions:** $P_i$ contains all private non-sensing actions of all agents, all public non-sensing actions that appeared in some trace of the team plan, and all the sensing actions of agent $i$.

**Transitions:** the transition function of public actions and agent $i$'s private actions is identical to that of the original problem definition. Private actions of other agents are determinized, leveraging our assumption about a desired effect for actions. The deterministic version of $a$ always achieves its desired effect. This relaxation is not essential, but reduces the difficulty in solving $P_i$.

**State and Observation Variables:** Observations of other agents do not appear in $P_i$. State variables that correspond only to the removed observations are also ignored, as they do not affect the transition functions of the actions in $P_i$. All other variables appear in $P_i$.

**Rewards:** (1) The reward for a private action is identical to its reward in the original Dec-POMDP and in $P_{team}$. (2) The reward $R(s,a)$ for a public action $a$ in state $s$ is defined as: $(i)$ if $s \models c$ for some context $c$ such that $\langle c,a\rangle \in CA_i$ $R(s,a)$ is positive. $(ii)$ If $s \not\models c'$ for any relaxed context $c'$ of some contexted action $\langle c,a\rangle \in CA_i$ then $R(s,a)$ is negative.

We use the following method for defining reward values:

1. Reward for an in-context action. We use the following steps:
   (a) Associate a value $R_\tau$ with each trace, reflecting the utility of the trace. Let $\tau$ be a trace, $R_\tau^+$ and $R_\tau^-$ the sum of positive and negative rewards, respectively, in $\tau$.
   (b) Distribute $R_\tau^+$ among the agents based on their relative contribution to attaining this utility, yielding $R_\tau^i$ for each $i$. Let $R_\tau^{-,i}$ be the total negative reward in $\tau$ associated with agent $i$'s actions only (including collaborative actions). Define $R_\tau^i = R_\tau^+ \cdot \frac{R_\tau^{-,i}}{R_\tau^-}$. This is the relative portion of reward we want to allocate to agent $i$'s actions in the trace.
   (c) For each agent and trace, distribute $R_\tau^i$ to each instance $e$ of the agent's public actions in this trace, yielding $R_{\tau,e}^i$. Let $e_j = (s_j, a_j, s'_j, o_j, r_j)$ be the $j^{th}$ step in trace $\tau$. We distinguish between contributing and non-contributing steps (defined below). If $e_j$ is non-contributing then $R_{\tau,e}^i = 0$. Otherwise, $R_{\tau,e}^i$ is defined using the following process:
      i. Associate a cost $c_e$ with $e$ (defined below).
      ii. Compute the relative weight, $w_e$, of step $e$ in $\tau$: $w_e = \frac{c_e}{R_\tau^{-,i}}$.
      iii. Define $R_{\tau,e}^i = w_e \cdot R_\tau^i$.
   (d) Associate with $\langle c,a\rangle$ the average value $r_{c,a}$ of $R_{\tau,e}^i$ over all traces $\tau \in \mathrm{T}$, and all steps $e \in \tau$ such that $e$ involves the execution of $a$ in a state satisfying $c$. The reward assigned to a CA in the model is the average reward $r_{CA}$ of steps in which it appears in the traces. Formally:

$$r_{CA} = \frac{\sum_{\tau \in \mathrm{T}} \sum_{e \in \tau \wedge proj_i(e) = CA} R_{\tau,e}^i}{|\{e | e \in \tau, \tau \in \mathrm{T}, proj_i(e) = CA\}|} \tag{1}$$

   $proj_i(e)$ is the contexted action obtained when projecting the state and action of $e$ w.r.t. agent $i$.
2. Penalty for an out-of relaxed-context action. We associate with the execution of an action in a state that does not satisfy any of its *relaxed* contexts, a negative reward $-\max_{ca \in CA_i} r_{ca} \cdot |CA_i|$, an upper bound on the sum of rewards that can be achieved from applying contexted actions.

To complete the description above, step $e_j = (s_j, a_j, s'_j, o_j, r_j)$ is a *contributing* step of agent $i$, if $a_j$ contains an action of agent $i$, and either (i) $s_j$ did not

appear earlier in the trace, or (ii) $r_j > 0$, i.e., a positive reward was gained. To define $c_e$, we iterate over the steps in $\tau$, accumulating the step costs (negative rewards). When encountering a contributing step of agent $i$, we assign it with the accumulated cost, and reset the accumulation.

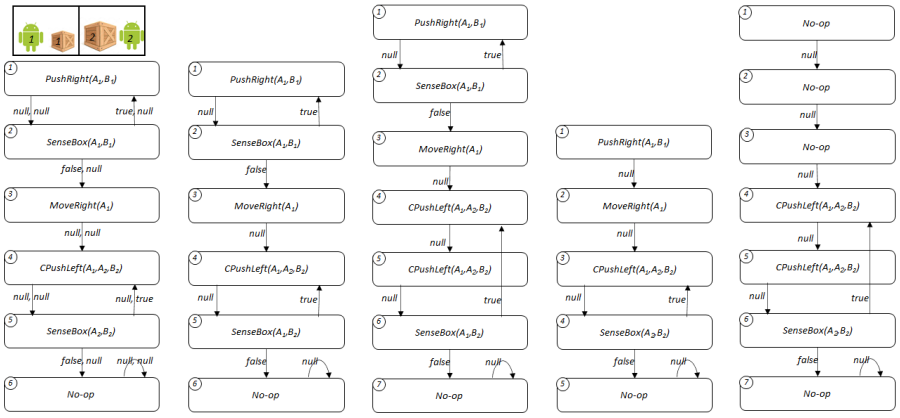
*Example 5.* We now construct $A_1$'s single-agent problem. We denote the CAs from the previous example by $ca_1, ca_2$, and their reward with $r_{ca_1}, r_{ca_2}$. We follow the projection stages one by one: (1) Push actions are the only public actions and we leave only the ones observed in traces: $\mathit{PushRight}(A_1, B_1)$, $\mathit{CPushLeft}(A_1, B_2)$, $\mathit{CPushLeft}(A_2, B_2)$. All other Push actions are removed. Notice that we keep $A_2$'s $\mathit{CPushLeft}(A_2, B_2)$ as $A_1$ might need to simulate it. (2) We remove the two sensing actions of $A_2$ and its observation variables $\omega_1^2, \omega_2^2$. (3) We leave all private actions of both agents. They are deterministic to start with, so no change is needed. (4) We set rewards $r_{ca_1}, r_{ca_2}$ to $ca_1$ and $ca_2$ respectively. (5) We set a penalty of $-2 \cdot max(r_{ca_1}, r_{ca_2})$ to the remaining public actions, applied in any context except for the CA's *relaxed* contexts. The relaxed context for $A_1$'s CAs is: $\{\langle\langle X_{A1} = L\rangle, \mathit{PushRight}(A_1, B_1)\rangle, \langle\langle X_{A1} = R\rangle, \mathit{CPushLeft}(A_1, B_2)\rangle\}$. (6) The reward for pushing the boxes to the target cells is set to 0, as we reward the agent only for doing its public actions in context.

### 3.2 Policy Adjustment and Alignment

We now solve the agent specific POMDPs $P_i$ and obtain agent specific policies $\pi'_i$, in the form of a policy graph for each agent. These policy graphs may contain private actions of other agents, and are likely not well synchronized. For example, there may be a low probability that collaborative actions are executed jointly at the same time. We now adjust the policies to obtain better synchronization.

First, we remove the private actions of all other agents from $i$'s policy graph, introduced to "simulate" the behavior of other agents in $P_i$. Next, we attempt to align policies to increase the probability that actions of different agents occur in the same order as in the team plan. An action $a_1$ of agent 1 that sets the context value of a variable in the context of action $a_2$ of agent 2 should be executed before $a_2$. Collaborative actions should be executed at the same time by all participating agents. As each policy is local to an agent, and action effects are stochastic, one cannot guarantee perfect synchronization. However, using a few heuristics, we attempt to increase the probability of synchronization.

For each public action $a$ in an agent's policy graph we select all simple paths from the root to $a$, and map them to their *identifiers*, where an *identifier* of a path is the sequence of public actions along it. Our goal is to equalize execution time of public actions occurrences with a shared identifier. For a public action $a$ with a shared identifier in multiple agents' graphs: let $l$ be the length of the longest simple path to the action in all relevant policy graphs, including private actions. In any graph where the length is less than $l$, we add *no-op* actions prior to $a$ to delay its execution. We use an iterative process — we begin with the action with the shortest identifier (breaking ties arbitrarily), and delay its execution where needed using *no-op*s. Then, we move to the next action, and so




(a) $\pi_{team}$ (b) $\pi'_1$ (c) $\pi_1$ (d) $\pi'_2$ (e) $\pi_2$

Fig. 1: A 2-cell box pushing problem, and the resulting plan graphs. The agents must switch between the boxes. Box $B_1$ is light, $B_2$ is heavy and must be pushed by the two agents jointly.

forth. After the alignment, we replace in each agent's aligned policy graph all actions of other agents with a *no-op*.

Finally, we handle the problem of a potential "livelock" between *collaborative* actions. Consider a scenario where two agents need to perform a non-deterministic collaborative action whose effect can be directly sensed. Each agent executes its part of the action and then senses whether it succeeded. Due to the stochastic nature of previous actions, one agent may execute the collaborative action one step before the other. In that case, it will sense its failure and repeat, with the second agent acting identically, one step later. To handle this, given a collaborative action with $n$ collaborating agents, we modify the graph so that every collaborative action that is part of a cycle is repeated by every agent for $n$ times instead of just once, preventing this livelock. This may be non-beneficial if a cost is associated with a collaborative action. To decide whether to apply it, during the reward definition stage, for each CA that contains a collaborative action, we calculate the expected reward of repeating the action, and sum it over all CAs. If the sum is positive, we enable the method.

*Example 6.* Figures 1b, 1d show plan graphs generated from the agent POMDPs. Here, edges are labeled by single-agent observations. In $\pi'_1$, the sensing action allocated to $A_2$ in $\pi_{team}$ is replaced by a sensing action of $A_1$ (node 5). In $\pi'_2$ appear the simulated actions of $A_1$ (nodes 1 and 2). Figures 1c, 1e show the policy graphs after the alignment and adjustments procedure. $\pi_1$ shows the repeated collaborative push action for live-lock avoidance (nodes 4 and 5). In $\pi_2$ the simulated actions of $A_1$ are replaced by *no-op* (nodes 1 and 2), and another *no-op* is added for alignment with $\pi_1$ (node 3).

## 4 Empirical Evaluation

We evaluate our algorithm on 3 domains: two variations of the popular cooperative Box Pushing, Dec-Tiger and Rock Sample. TIS uses SARSOP [9] as the underlying POMDP solver.

We compare TIS, with 3 Dec-POMDP solvers, GMAA*-ICE [13], JESP [10], and DICEPS [12] using MADP-tools [14]. The GMAA*-ICE solver provides optimality guarantees. JESP searches in policy space, performing alternating maximizations using dynamic programming. DICEPS is an approximate method, which does not excel on smaller domains, but presumably can scale to larger domains.[2] We also compare TIS to 3 state-of-the-art MARL algorithms: two versions of WQMix [16], and QTRAN [18] All experiments were conducted on a PC with Intel-core i7-5500U CPU @ 2.40GHz with 4 cores and 7.7GB of memory. TIS, GMMA-ICE, and JESP were run under Windows-11 and the others were run under Ubuntu 20.04.1.

GMAA*-ICE and DP-JESP require an horizon specification, specified under column $H$. TIS computes a policy for an unbounded horizon and its $H$-value specifies the average number of steps until reaching the goal state. DICEPS uses restarts, and repeatedly returned an overflow error. To generate comparable running times to TIS, we rerun it multiple time, and used the maximal score over these runs, which is equivalent to simply letting it run longer.

For GMAA*-ICE and DP-JESP we report the computed policy value. For TIS the *value* column provides the average discounted accumulated reward over 1000 simulations truncated at the relevant horizon. The *avg* column denotes average number of steps for all agents to reach the goal state. For MARL algorithms, we measure maximum average discounted reward over a number of test runs. The discount factor was set to $\gamma = 0.99$.

Planners were given 1 hour to solve each $\langle$*configuration, horizon*$\rangle$ pair. MARL algorithms were given longer times. We report running times in seconds.

### 4.1 Domains

**Box Pushing:** agents on a grid must push boxes to their destinations [6, 5]. Light boxes can be pushed by a single agent. Heavy boxes must be pushed collaboratively by two agents. Agents can sense for a box at the present location. The initial box locations are uncertain. We also consider a variant of this domain in which we add a penalty when a push action is performed in a cell with no box. Problem names are composed of 5 elements, specifying width, height, number of agents, number of light boxes, and number of heavy boxes.

**Dec-Tiger:** Agents must open a door avoiding a tiger [11]. The tiger's location resets following a door opening. Collaboratively opening a tiger door results in a lower penalty compared to single agent opening actions. Agents have noisy

[2] DICEPS, while not new, was recommended to us, independently, by two senior researchers as still being a state-of-the-art approximate solver.

Table 2: Results for Collaborative Dec-Tiger on state-of-the-art Dec-POMDP and MARL solvers and TIS. Best overall value for each problem in bold.

| Collaborative Dec-Tiger, $\|S\| = 8, \|A\| = 5, I = \langle (0,1,0),(0,1,1) \rangle$ | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H | DP-JESP | | GMAA*-ICE | | DICESP | | OW QMix | | QTran | | TIS | |
| | Time | Value | Time | Value | Time | Value | 300s | 3600s | 300s | 3600s | Time | Value |
| 3 | 0.42 | 1.96 | 0.24 | **11.15** | 121.54 | 0 | -7.84 | 7.44 | 0 | 2.55 | 6.03 | 11.10 |
| 4 | 33.15 | 10.05 | 841.67 | 11.03 | 155.98 | 3.2 | -7.87 | **13.602** | 0 | 0.958 | " | 9.12 |
| 5 | 1792.22 | 5.78 | × | - | 206.56 | 4.03 | 1.68 | **14.454** | 0 | 4.08 | " | 8.56 |
| 6 | × | - | × | - | 243.07 | 5 | 2.61 | **15.473** | 0 | 1.96 | " | 9.05 |
| 10 | × | - | × | - | x | x | -17.79 | **18.136** | 0 | 0.758 | " | 10.19 |
| 20 | × | - | × | - | x | x | -13.91 | **20.058** | 0 | 1.79 | " | 16.42 |
| 30 | × | - | × | - | x | x | -19.43 | 18.954 | 0.58 | **54.907** | " | 20.10 |
| 40 | × | - | × | - | x | x | -11.702 | **26.128** | 0 | 10.49 | " | 24.62 |

observations on the tiger's location. We use a larger Dec-Tiger version that requires agents to move on a grid [1].

**Decentralized Rock-Sample:** We suggest a new variant of Mars exploration problem [17], where two rovers move on a grid that contains rocks that must be sampled . The grid is divided into overlapping control areas, one per rover. The rovers can move around the grid, sense the rocks' quality, which is initially unknown, and sample them. Agents are rewarded for sampling all the good rocks in their control area, but penalized directly for sampling a bad quality rock. Once a good quality rock is sampled, it turns bad. Rovers have a long range sensor, whose quality deteriorates with increased distance from the rock. Problem names are composed of the grid size and the number of rocks.

These problems call for solutions of different types. In Collaborative Dec-Tiger, the problem resets after one of the doors is opened and so the planning horizon can be short, there are no interdependent actions (i.e., ones setting up conditions for the execution of others), but tight coordination in the execution of door opening actions is desirable. Collaborative Box-Pushing rewards the agents for pushing a box to the target tile, requiring aggregating a chain of properly-directed pushes of a box to a single reward. With light boxes, a simple plan can use a single agent. Yet, efficient plans make use of the different agent positions to reduce (costly) move actions. With heavy boxes, agents must push the box at the same time, requiring even better coordination. The second box-pushing variant calls for trading off the cost of additional sensing with the penalty for moving a non-existent box. Decentralized Rock-Sampling rewards an agent only for achieving all of its objectives, which makes large-horizon planning ability crucial, giving no reward at all to partial solutions.

### 4.2 Results

Table 2 describes the results for Dec-Tiger. GMAA*-ICE, which is an optimal solver, produces the best policy for the shortest horizon, but cannot scale beyond

Table 3: TIS vs. Dec-POMDP solvers and vs. MARL solvers on Box-Pushing (BP) and Rock-Sample (DRS). Best value per problem in bold.

| Collaborative Box-Pushing | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Problem | | | DP-JESP | | | GMAA*-ICE | | | DICEPS | | | TIS | | | |
| Name | $\|S\|$ | $\|A\|$ | H | Time | Value | H | Time | Value | H | Time | Value | Avg | Time | Value | |
| 3,1,2,1,1 | 81 | 16 | 4 | 1861.30 | 279 | 4 | 30.23 | 330 | 4 | 221.71 | 0 | 15 | 6.07 | 613 | |
| 2,2,2,0,2 | 256 | 225 | 3 | 267.24 | 271 | 3 | 160.18 | 320 | 3 | 348.56 | 0 | 9 | 7.08 | **348** | |
| 2,2,2,0,3 | 1024 | 400 | 2 | 59.06 | 0 | 2 | 1053.27 | 414 | 2 | 514.09 | 0 | 17 | 125.50 | **514** | |
| 1-Penalty | 81 | 16 | 3 | 25.95 | 0 | 4 | 79.28 | 265 | - | - | - | 15 | 8.54 | 587 | |
| 2-Penalty | 256 | 225 | 3 | 495.61 | 135 | 3 | 446.03 | 214 | - | - | - | 9 | 11.66 | **354** | |
| 3-Penalty | 1024 | 400 | 2 | 38.70 | 0 | 2 | 1054.5 | 327 | - | - | - | 15 | 31.09 | **510** | |
| | | | OW QMIX | | | CW QMIX | | | QTRAN | | | TIS | | | |
| | | | H | 300s | 7200s | H | 300s | 7200s | H | 300s | 7200s | Avg | Time | Value | |
| 3,1,2,1,1 | 81 | 16 | 20 | 120.27 | 1165 | 20 | -8.91 | **1177** | 20 | 345.17 | 348 | 15 | 6.07 | **614** | |
| 2,2,2,0,2 | 256 | 225 | 15 | -3.57 | -1.98 | 15 | 0 | 0 | 15 | 0 | -0.79 | 9 | 7.08 | **349** | |
| 2,2,2,0,3 | 1024 | 400 | 20 | -2 | 0 | 20 | 0 | -2 | 20 | -0.8 | 0 | 17 | 125.50 | **514** | |
| 1-Penalty | 81 | 16 | 20 | -36.32 | 1144 | 20 | 76.97 | **1191** | 20 | -77.94 | 372 | 15 | 8.54 | **587** | |
| 2-Penalty | 256 | 225 | 15 | 0 | -3.9 | 15 | 0 | 0 | 15 | 0 | 0 | 9 | 11.66 | **354** | |
| 3-Penalty | 1024 | 400 | 20 | 0 | 0 | 20 | 0 | 0 | 20 | -35.62 | -29 | 15 | 31.09 | **510** | |
| 3,2,3,0,2 | 7776 | 3375 | 20 | 0 | 0 | 20 | 0 | - | 20 | 0 | 0 | 12 | 89.83 | **276** | |
| 3,2,3,0,3 | 46656 | 8000 | 20 | 0 | 0 | 20 | 0 | 0 | 20 | 0 | 0 | 18 | 2210.28 | **406** | |
| 3,3,2,2,1 | 59049 | 324 | 20 | -4.37 | -2 | 20 | 0 | 0 | 20 | -72.76 | 0 | 39 | 3014.04 | **322** | |
| Decentralized Rock-Sampling | | | | | | | | | | | | | | | |
| Problem | | | DP-JESP | | | GMAA*-ICE | | | DICEPS | | | TIS | | | |
| 12,3 | 512 | 90 | 3 | 314.35 | 224 | 3 | 82.86 | 739 | 3 | 2018 | 755 | 18 | 1725.80 | **1028** | |
| 12,4 | 1024 | 100 | 3 | 1081.41 | 518 | 3 | 2867.06 | 508 | 3 | 2763 | 495 | 20 | 2438.82 | **1048** | |
| 20,4 | 2304 | 100 | 3 | 1838.84 | 111 | 3 | - | - | 3 | 2868 | 514 | 16 | 2434.88 | **1158** | |
| | | | OW QMIX | | | CW QMIX | | | QTRAN | | | TIS | | | |
| | | | H | 300s | 7200s | H | 300s | 7200s | H | 300s | 7200s | Avg | Time | Value | |
| 12,3 | 512 | 90 | 25 | -56.51 | 474 | 25 | -54.42 | 509 | 25 | -34.28 | 609 | 18 | 1725.80 | **1028** | |
| 12,4 | 1024 | 100 | 25 | -45.6 | 198 | 25 | -12.00 | 203 | 25 | 0 | 617 | 20 | 2438.82 | **1048** | |
| 20,4 | 2304 | 100 | 25 | -55.20 | 452 | 25 | -45.63 | 219 | 25 | -40.05 | 327 | 16 | 2434.88 | **1158** | |
| 20,6 | 9216 | 143 | 25 | -19.34 | 135 | 25 | -43.85 | 20 | 25 | -21.83 | 408 | 21 | 2537.10 | **1121** | |
| 28,6 | 16384 | 143 | 25 | -29.23 | 47 | 25 | 91.78 | 24 | 25 | -1.99 | 0 | 23 | 3310.95 | **1046** | |

4 steps. JESP scales to horizon 5, and DICEPS to horizon 6, but they produce policies of much lower quality than TIS, which can handle horizon 40 in about 6 seconds. DICESP, which should be able to scale well, terminated with an error after roughly 10-20 seconds for horizons 10 and higher. For the MARL algorithms we show results for two running times: 300 and 3600 seconds. In this domain, OW-QMix was able to perform better than TIS, but only when given 3600 seconds, as opposed to 6 for TIS. In the horizon 30 case, QTran was able to perform much better than TIS, but performed far worse on other horizons. Except for this case, TIS performed comparatively to, or better than other MARL algorithms but required much less time. As can be seen from the

Table 4: Comparing to optimal on small box pushing with $|S| = 8$ and $|A| = 16$. *Max* row is for maximal value for any horizon. ArgMax Horizon in parenthesis.

| | GMAA*-ICE | | TIS | |
|---|---|---|---|---|
| H | Time | Value | Time | Value |
| 4 | 1.15 | **426.91** | 3.40 | 306.62 |
| 5 | 2.09 | **438.34** | " | 301.56 |
| 6 | 6.97 | **448.19** | " | 357.44 |
| 7 | 8.98 | **450.97** | " | 412.54 |
| Max | 17.1 | **454.70** (25) | 3.40 | 412.54 (7) |

results for 300 seconds, MARL algorithms cannot compete with TIS as far a policy quality with shorter running times, even when given 50X running time. CW-QMix was always dominated by OW-QMix and is therefore omitted.

Table 3 shows results for Box Pushing and Rock Sample. Both DP-JESP and GMAA*-ICE could not handle horizons larger than 3, while TIS can consider much longer action sequences, and, as such, collect much higher rewards. As noted above, Decentralized Rock-Sampling requires much lengthier horizons to achieve any reward, and TIS's advantage here is pretty clear. Among the MARL algorithms, QMix was able to produce much better results on one problem instance (again, requiring orders of magnitude more time), but MARL solvers failed on harder Box Pushing domains that contain more heavy boxes that require a collaborative push action. Similar results were obtained with the alternative reward function, punishing attempts to push a non-existent box. TIS is consistently better and faster than the MARL solvers on Rock Sample, again due to MARL difficulty in learning policies that require more complex coordination.

In the Box Pushing and Rock Sample we can also test the scalability of TIS. These are domains that current Dec-POMDP solvers cannot handle, and hence we only provide a comparison with MARL algorithms. As can be seen, TIS can consider very long horizons even in these significantly larger problems. The size of the hardest configuration, BP-33221, approaches the maximal problems that the single agent POMDP solver SARSOP can solve, indicating that TIS could scale to even larger problems given a stronger POMDP solver. We are not aware of any other Dec-POMDP solver that is able to approach state sizes even close to these on these domains: over 59000 in Box Pushing and over 16000 in Rock Sample. The MARL solvers were unable to generate reasonable solution within 7200 seconds for these larger problems. (For this reason, we do not provide results for shorter running times). To complete the picture, we note that in the hardest Box-Pushing and Rock-Sample instances, TIS was able to achieve the goal in 100% of all trials for Box Pushing and in 97% of all trials for Rock Sample.

TIS contains many heuristic choices that may cause it to obtain sub-optimal policies. To measure this, Table 4 focuses on a small box pushing problem that GMAA*-ICE can solve optimally. We see that TIS manages to produce reasonable results that are about 10% worse on the larger horizons. Furthermore, in Dec-Tiger, which calls for strong agent synchronization, TIS does virtually

the same as GMAA*-ICE on horizon 3. On medium size horizons TIS performs worse, probably because there is more opportunity for unsynchronized actions.

### 4.3 Discussion

As we observed above, over the 3 domains that we experiment with, TIS is substantially better than all other solvers. It produces policies which are close to optimal on smaller problems with shorter horizons, and scales many orders of magnitude beyond existing Dec-POMDP solvers.

In comparison to state-of-the-art MARL algorithms, TIS is much faster, and often returns much better policies. Of course, MARL algorithms do not receive the model as input and must compensate for it by running many trials. Nevertheless, we provide this comparison for two reasons. First, there is a perception that RL algorithms are a magic bullet and are the state-of-the-art for stochastic sequential decision making. Second, model-free RL algorithms are often suggested as an alternative to model-based solvers on domains with large state spaces, as they do not need to maintain and query an explicit model, which in larger problems can be difficult to represent. While this is certainly true, the time and resources required to compensate for the lack of a model can be substantial. As we have shown here, domains that require better coordination are still challenging for state-of-the-art MARL solvers. In longer (non-exhaustive) experiments conducted on QMIX, the results on these domains did not improve even given over 10 hours. Thus, we think that it can be safely concluded that when a model is available and a plan is needed quickly, TIS is currently the best option.

Finally, we briefly discuss the MCEM Dec-POMDP solver [19]. This algorithm is an example of an approximate solver that can scale to huge problems. Indeed, MCEM was able to solve a grid-based traffic problem with 2500 agents and roughly $10^{100}$ states, which is certainly remarkable. MCEM excels on this domain because it is very loosely coupled, and a simple local controller per agent can generate good behavior. MCEM exploits these domain properties to truly factor the problem (aided by a hand-coded MDP policy for policy exploration).

TIS cannot handle such a domain because the team problem would be too large to describe formally, and no POMDP solver can scale to such problem sizes. On the other hand, [19] also tested MCEM on standard Dec-POMDP benchmarks that do not enjoy such weak coupling, and in which stronger implicit coordination between the agents' actions is required. In these domains, MCEM was unable to scale to the problems sizes TIS handles. For example, the largest box pushing problem solved by [19] had 100 states, and the largest Mars Rover problem solved (which is quite similar to rock-sample) had 256 states compared with 59K and 16K states, respectively, for TIS, limited only by the capabilities of the underlying POMDP solver.

## 5 Conclusion

TIS is a general approach for solving Dec-POMDPs. We described a particular implementation that solves Dec-POMDP by solving multiple POMDPs. First,

we solve a team POMDP – a single-agent relaxation of the Dec-POMDP. Then, we solve an imitation POMDP for each agent that seeks to generate a policy that imitates that agent's behavior in the team policy. Finally, policy graphs are aligned to improve synchronization. We report promising empirical results. Our implementation of TIS solves significantly larger problems and horizons than existing approximate Dec-POMDP solvers on standard benchmarks in which the problem is not very loosely coupled. It compares well to near-optimal solvers on the problems they can solve, and is much better than MARL algorithms on domains that require some agent coordination.

While the high level flow of TIS is attractive, the particular implementation of the steps is often very specific, not unlike many RL/MARL/DL approaches. In particular, our current synchronization step relies heavily on no-op insertion. The ability to use no-ops implies that the agents are the sole cause of change in the environment. Yet, one exciting aspect of the TIS schema is its generality, and the many exciting opportunities it offers for instantiating each element, in more general and more effective ways.

## References

1. Amato, C., Bernstein, D.S., Zilberstein, S.: Optimizing fixed-size stochastic controllers for pomdps and decentralized pomdps. JAAMAS **21**(3), 293–320 (2010)
2. Bazinin, S., Shani, G.: Iterative planning for deterministic qdec-pomdps. In: GCAI-2018. 4th Global Conference on Artificial Intelligence. vol. 55, pp. 15–28 (2018)
3. Bernstein, D.S., Givan, R., Immerman, N., Zilberstein, S.: The complexity of decentralized control of markov decision processes. Mathematics of operations research **27**(4), 819–840 (2002)
4. Boutilier, C., Dean, T., Hanks, S.: Decision-theoretic planning: Structural assumptions and computational leverage. J. Artif. Int. Res. **11**(1), 1–94 (1999)
5. Brafman, R.I., Shani, G., Zilberstein, S.: Qualitative planning under partial observability in multi-agent domains. In: AAAI'13 (2013)
6. Carlin, A., Zilberstein, S.: Value-based observation compression for dec-pomdps. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1. pp. 501–508 (2008)
7. Dietterich, T.G.: Hierarchical reinforcement learning with the maxq value function decomposition. JOURNAL OF AI RESEARCH **13**, 227–303 (2000)
8. Hausknecht, M.J., Stone, P.: Deep recurrent q-learning for partially observable mdps. In: 2015 AAAI Fall Symposium. pp. 29–37. AAAI Press (2015)
9. Kurniawati, H., Hsu, D., Lee, W.S.: Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In: In Proc. Robotics: Science and Systems (2008)
10. Nair, R., Tambe, M., Yokoo, M., Pynadath, D., Marsella, S.: Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In: IJCAI'03. pp. 705–711 (2003)
11. Nair, R., Tambe, M., Yokoo, M., Pynadath, D., Marsella, S.: Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In: IJCAI. vol. 3, pp. 705–711 (2003)
12. Oliehoek, F., Kooij, J., Vlassis, N.: The cross-entropy method for policy search in decentralized pomdps. Informatica **32**, 341–357 (01 2008)

13. Oliehoek, F.A., Spaan, M.T.J., Amato, C., Whiteson, S.: Incremental clustering and expansion for faster optimal planning in decentralized POMDPs. JAIR **46**, 449–509 (2013)
14. Oliehoek, F.A., Spaan, M.T.J., Terwijn, B., Robbel, P., Messias, J.a.V.: The madp toolbox: An open source library for planning and learning in (multi-)agent systems. J. Mach. Learn. Res. **18**(1), 3112–3116 (Jan 2017)
15. Oliehoek, F.A., Spaan, M.T.J., Whiteson, S., Vlassis, N.: Exploiting locality of interaction in factored dec-pomdps. In: AAMAS. p. 517–524 (2008)
16. Rashid, T., Farquhar, G., Peng, B., Whiteson, S.: Weighted QMIX: expanding monotonic value function factorisation for deep multi-agent reinforcement learning. In: Advances in Neural Information Processing Systems (2020)
17. Smith, T., Simmons, R.: Point-based pomdp algorithms: improved analysis and implementation. In: UAI. pp. 542–549 (2005)
18. Son, K., Kim, D., Kang, W.J., Hostallero, D., Yi, Y.: QTRAN: learning to factorize with transformation for cooperative multi-agent reinforcement learning. In: ICML. pp. 5887–5896 (2019)
19. Wu, F., Zilberstein, S., Jennings, N.R.: Monte-carlo expectation maximization for decentralized pomdps. In: IJCAI. pp. 397–403 (2013)