A Factored Approach To Solving Dec-POMDPs

Eliran Abdoo, Ronen I Brafman, Guy Shani
Ben-Gurion University, Israel
eliranb,brafman,shanigu@bgu.ac.il

Abstract
Dec-POMDPs model planning problems under uncertainty and partial observability for a distributed team of cooperating agents planning together but executing their plans in a distributed manner. This problem is very challenging computationally (NEXP-Time Complete) and consequently, exact methods have difficulty scaling up. In this paper we present a heuristic approach for solving certain instances of Factored Dec-POMDP. Our approach reduces the joint planning problem to multiple single agent POMDP planning problems. First, we solve a centralized version of the Dec-POMDP, which we call the team problem, where agents have a shared belief state. Then, each agent individually plans to execute its part of the team plan. Finally, the different solutions are aligned to achieve synchronization. Using this approach we are able to solve larger Dec-POMDP problems, limited mainly by the abilities of the underlying POMDP solver.

1 Introduction
Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs) are a popular model for planning in stochastic environments under uncertainty by a distributed team of agents (Oliehoek and Amato 2016). In this model, a team of agents attempts to maximize the team’s cumulative reward where each agent has only partial information about the state of the system during execution. The team can plan together centrally prior to acting, but during execution each agent is aware of its own observations only. Communication is possible only through explicit communication actions, if these are available.

To achieve their common goal agents must coordinate their actions in two ways: First, as in single agent problems, actions must be coordinated sequentially. That is, current actions must help steer the system later towards states in which greater reward will be possible. For example, to be rewarded for shipping a product, it must first be assembled. Second, agents may need to coordinate their simultaneous actions because their effects are dependent, e.g., a heavy box can only be pushed if two agents push it simultaneously.

Our focus is on centralized off-line planning for distributed execution. That is, offline, a solver with access to the complete model must generate a policy for each agent. An agent’s policy specifies which action it executes as a function of the agent’s history of actions and observations. Such policies can be represented by a policy graph where nodes are labeled by actions, and edges are labeled by observations. Online, each agent executes its own policy independently of the other agents. The challenge is to generate policies that provide sufficient coordination, even though each agent makes different observations at run-time. Thus, agents’ beliefs over which states are possible are typically different.

Dec-POMDPs are notoriously hard to solve – they are NEXP-Time hard (Bernstein, Zilberstein, and Immerman 2013), implying that only the smallest toy problems are optimally solvable. However, many approximate methods for solving Dec-POMDPs have been proposed, with steady progress. Some of these methods generate solutions with bounds on their optimality (Oliehoek et al. 2013; Seuken and Zilberstein 2007; Oliehoek, Kooij, and Vlassis 2008), and some are heuristic in nature (Nair et al. 2003). However, current methods typically do not scale to state spaces with more than a few hundreds of states.

In this paper we describe a heuristic approach for solving Dec-POMDPs that scales to much larger state spaces. The key idea is to solve a Dec-POMDP by solving multiple POMDPs. First, we solve the team POMDP, a POMDP in which every observation by one agent is immediately available to the other agents. Hence, all agents have the same belief state. The solution of the team POMDP can be represented by a policy graph — the team policy graph. It provides us with a skeleton for the solution of the Dec-POMDP, specifying what each agent needs to provide for the team. Naturally, this policy is not executable by the agents, because agents cannot condition their actions on the observations of other agents in the real world.

Hence, in the next stage, we let each agent solve a POMDP in which it is rewarded for behaving following the specification in the team policy. This leads to the generation of a policy tree for each agent. These policy trees are often not well synchronized. In the last step we synchronize the policy trees by delaying the actions of agents to improve the probability of good coordination.

We implemented our algorithm and tested it on several configurations of a benchmark problem: Collaborative Box-Pushing which is a variation of the Cooperative Box Pushing
problem. We show that the algorithm manages to scale well beyond current Dec-POMDP solvers. One of the main properties of the domain, is that agents policies are only loosely coupled. That is, the need for actions that affect state components that are relevant to all agent, is sparse. That sparsity allows for each agent to independently construct a plan that consists mostly of its own private actions without requiring it to consider the other agents’ behavior. This allows us to achieve good decentralized policies even when achieving the goal requires many steps, compared to planning directly over the Dec-POMDP model.

2 Background
We now provide needed background on POMDPs, Dec-POMDPs, their factored representation, and policies. We also introduce the concept of private and public variables and actions in Dec-POMDPs.

2.1 POMDPs
A POMDP is a model for single-agent sequential decision making under uncertainty and partial observability. Formally, it is a tuple $P = (S, A, T, R, O, \gamma, h, b_0)$, where:

- $S$ is the set of states. The future is independent of the past, given the current state.
- $A$ is the set of actions. An action may modify the state and/or provide information about the current state.
- $T : S \times A \rightarrow \prod(S)$ is the state transition function. $T(s, a, s')$ is the probability of transitioning to $s'$ when applying $a$ in $s$.
- $R : S \times A \times S \rightarrow \mathbb{R}$ is the immediate reward function. $R(s, a, s')$ is the reward obtained after performing $a$ in $s$ and reaching $s'$.
- $\Omega$ is the set of observations. An observation is obtained by the agent following an action, and provides some information about the world.
- $O : S \times A \rightarrow \prod(\Omega)$ is the observation function, specifying the likelihood of sensing a specific observation following an action. $O(s', a, o)$ is the probability of observing $o \in \Omega$ when performing $a$ and reaching $s'$.
- $\gamma \in (0, 1)$ is the discount factor, quantifying the relative importance of immediate rewards vs. future rewards.
- $h$ is the planning horizon — the amount of actions that an agent executes before terminating. The horizon may be infinite.
- $b_0 \in \prod(S)$ is a distribution over $S$ specifying the probability distribution over the initial state.

For ease of representation, we assume that agent actions are either sensing actions or non-sensing actions. An agent that applies a non-sensing action receives the observation $null-obs$. In addition, we also assume that every action has an effect that we consider as the successful outcome, while all other effects are considered failures. We later explain how this assumption can be omitted in the relevant parts.

Often, the state space $S$ is structured, i.e., it consists of assignments to some set of variables $X_1, \ldots, X_k$, and the observation space $\Omega$ is also structured, consisting of a set of 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, $T$, $O$, and $R$ can be represented compactly by, e.g., a dynamic Bayesian network (Boutilier, Dean, and Hanks 1999). Formats such as RDDL (Sanner 2011) and POMDPX (POM 2014) exploit factored representations to specify POMDPs compactly.

Example 1. Consider a simple Box-Pushing in a 2 cell grid. The left cell is marked by $L$ and the right cell by $R$. The agent begins in the left cell. There is a single box, that starts in the right cell. The agent can either move, sense its current cell or push a box from its current cell. Both move and push can be done in any direction — left and right. The agent’s goal is to push the box to the right cell. The state is composed of 2 variables: the location of the agent and the location of the box. Each variable can take one of two values: $L$ or $R$. The sense action returns an observation telling whether there’s a box in the agent’s cell, while the move and push actions are non-sensing actions always returning $null-obs$. The push action has a success probability of 0.8.

A solution to a POMDP can be formed as a policy, assigning to each history of actions and observation ($AO$-history) the next action to execute. Such a policy is often represented using a policy tree or, more generally, a policy graph (also called a finite-state controller).

A policy graph $G = (V, E)$ is a directed simple graph, in which each vertex is associated with an action, and each edge is associated with an observation. For every edge $v \in V$ and every observation $o \in \Omega$ exactly one edge emanates from $v$ with the label $o$. The graph has a single root which acts as its entry point. Every $AO$-history $h$ can be associated with some path from the root to some vertex $v$, and the action labelling $v$ is the action that the policy associates with $h$.

Finally, using a policy graph to direct the agent on the problem produces a trace — an execution trajectory. A trace $T$ of length $l$ is a sequence of quintuplets $e_i = (s_i, a_i, s'_i, o_i, r_i)$, namely steps, that occurred during a possible policy execution where: $s_i$ is a state in step $i$ and $s_0$ is the initial state; $a_i$ is the action taken in step $i$; $s'_i$ is the result of applying $a_i$ in $s_i$; $o_i$ is the observation received after taking $a_i$ and reaching $s'_i$; and is the reward received for taking the $a_i$ in $s_i$ and reaching $s'_i$. Clearly, $\forall i$ such that $0 \leq i < l-1$, we have $s'_i = s_{i+1}$.

2.2 Dec-POMDP
A Dec-POMDP models problems where there are $n > 1$ acting agents. These agents are part of a team, sharing the same reward, but they act in a distributed manner, sensing different observations. Thus, their information state is often different. Formally, a Dec-POMDP for $n$ agents is a tuple $P = (S, A = \bigcup_{i=1}^{n} \{A_i\}, T, R, \Omega = \bigcup_{i=1}^{n} \{\Omega_i\}, O, \gamma, h, b_0)$, where:

- $S, \gamma, h, b_0$ are defined as in a POMDP.
- $A_i$ is the set of actions available to agent $i$. We assume that $A_i$ contains a special $no-op$ action, which does not change the state of the world, and does not provide any informative observation. $A = A_1 \times A_2 \times \cdots \times A_n$ is the
set of joint actions. On every step each agent $i$ chooses an action $a_i \in A_i$ to execute, and all agents execute their actions jointly. $\langle a_1, \ldots, a_n \rangle$ is known as a joint action. We often treat the single-agent action $a_i$ as a joint-action, with the understanding that it refers to the joint-action $\langle \text{no-op}, \ldots, a_i, \ldots, \text{no-op} \rangle$.

- $T : S \times A \rightarrow \prod_i T_i(S)$ is the transition function. Transitions are specified for joint actions, that is, $T_i(s, \langle a_1, \ldots, a_n \rangle, s')$ is the probability of transitioning from state $s$ to state $s'$ when each agent $i$ executes action $a_i$.
- $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function. Rewards are also specified over joint actions.
- $\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_n$ is the set of joint observations. Each $\Omega_i$ contains a special null-obs observation received when applying a non-sensing action.
- $O : S \times A \rightarrow \prod_i O_i(s_i)$ is the observation function, specified over joint actions. $O_i(s'_{i}, \langle a_1, \ldots, a_n \rangle, \langle o_1, \ldots, o_n \rangle)$ is the probability that when all agents execute $\langle a_1, \ldots, a_n \rangle$ jointly and reach state $s'$, each agent $i$ observes $o_i$.
- $\gamma$ is the discount factor.
- $h$ is the horizon.
- $b_0 \in \prod (S)$ is a distribution over $S$ specifying the probability that each agent begin its execution in each state. In principle, different agents may have different initial belief states, but we make the (common) assumption that the initial belief state is identical.

Example 2. We extend the previous example to a Dec-POMDP by adding an agent at the right cell and a second box that starts in the left cell. The agents are denoted by Agent1 and Agent2 and the boxes by Box1 and Box2. Box1 must reach the left cell, and Box2 must reach the right cell.

As in the case of POMDPs, Dec-POMDPs can also be represented in a factored manner (Oliehoek et al. 2008), although most work to date uses the flat-state representation. We add the notion of observation variables, which capture the observation value each agent obtains following an action. Each observation variable is denoted by $\omega_i$, which takes values in $\Omega_i$, and represents the observation of agent $i$.

Example 3. In our example, the state is now composed of 4 state variables: the location of each box – $\langle X_{B1}, X_{B2} \rangle$ – and the location of each agent – $\langle X_{A1}, X_{A2} \rangle$. In addition, there are two observation variables – $\langle \omega_1, \omega_2 \rangle$.

An important element of a factored specification of Dec-POMDPs is a compact formalism for specifying joint-actions. If there are $|A|$ actions in the domain, then, in principle, there are $O(|A|^n)$ possible joint actions. Specifying all joint actions explicitly is unrealistic for large domains.

In many problems of interest we may expect that most actions will not interact with each other. A pair of actions $a \in A_i$, $a' \in A_j$ is said to be non interacting, if their effects when applied jointly (in the same joint action) is the union of their effects when applied separately. 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 refer to as collaborative actions (Bazinin and Shani 2018). For a more detailed discussion of the compact specification of joint-actions, see (Shekhar and Brafman 2020).

Example 4. We alter our example further by introducing a collaborative action. To do so, we need to convert one of the boxes to a "heavy" box - a box that requires both agents to push it. We will convert Box1 to such box. Both agents now have also the option to apply a collaborative-push action in any specified direction. If both agents apply that action to push Box1 while in the same cell with it, the box will transit.

Finally, a solution to a Dec-POMDP is a set of policies $\rho_i$, one for each agent. It maps action-observation sequences of this agent to actions in $A_i$. As in POMDPs, these policies can be specified using a policy graph for each agent. The policy graph for agent $i$ associates nodes with actions in $A_i$, and edges with observations in $\Omega_i$.

2.3 Public and Private Actions and Variables

We find it useful to define the concept of public and private variables and actions. State variables can influence or be influenced by an agent’s action. State variables that are influenced by several agents are called public and state variables that are influenced only by a single agent are private.

The concept of private and public (or local and global) variables (Brafman and Domshlak 2006) has been used extensively in work on privacy-preserving multi-agent planning (e.g., (Nissim and Brafman 2014; Maliah, Brafman, and Shani 2017)) and, more recently, in work on solving qualitative variants of Dec-POMDPs (Brafman, Shani, and Zilberstein 2013; Shekhar, Brafman, and Shani 2019).

We now explain how we extend these concepts to factored Dec-POMDPs. These definitions are based on the notions of preconditions and effects, as used in classical planning. Let $a \in A_i$ be a non-sensing action of agent $i$. We identify $a$ with the joint action $\langle \text{no-op}, \ldots, a, \ldots, \text{no-op} \rangle$. We say that a state variable $X_i$ is an effect of $a$ if there is some state $s$ for which there is a positive probability that the value of $X_i$ changes following $a$. We denote the effects of $a$ by $\text{eff}(a)$.

We say that state variable $X_i$ is an influencer of $a$ if $a$ behaves differently for 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 $O(s_1, a, o) \neq O(s_2, a, o)$ for some observation $o$. We denote influencers of $a$ by $\text{inf}(a)$. We refer to the union of the influencers and effects of $a$ as the relevant variables of $a$, denoted $\text{rel}(a)$.

If $a$ is a sensing action, we define $\text{inf}(a)$ similarly, i.e., $X_i \in \text{inf}(a)$ if there are two states $s_1, s_2$ that differ only in the value of $X_i$ such that the distribution over the values of the observation variable of the agent performing $a$ at $s_1$ and at $s_2$ are different.

For collaborative actions, the definitions remain the same except that now we identify $a$ with the the joint-action that is composed of the actions of the collaborating agents, and no-ops for the rest.

We say that a variable $X_i$ is relevant to agent $j$, if $X_i$ is relevant to some some $a \in A_j$. Finally, $X_i$ is public if it is relevant to more than one agent, and it is private otherwise.
Example 5. In our running example, $X_{B1}$ and $X_{B2}$ are both public variables, as they are relevant of both agents’ push actions. $X_{A1}$, $X_{A2}$ are private variables of agent 1 and agent 2 respectively, as they are the relevant only to each respective agent’s move action. The same holds for $\omega_1$ and $\omega_2$ with respect to the sense actions. Furthermore, the move and sense actions are private actions, while the push actions are public.

3 FDMAP - Factorized Distributed MAP

Given the input factored Dec-POMDP problem $P$, we first generate the team POMDP $P_{\text{team}}$. We solve $P_{\text{team}}$ using an off-the-shelf POMDP solver – we used SARSOP (Kurniawati, Hsu, and Lee 2008) – and output the team policy.

Next, we then use the team policy to produce traces, which are simulations of the team policy over the team problem. Using the traces, we project the team problem with respect to each agent, as follows: First, for each agent, we extract from the traces a set of public actions and the context in which they were applied, which we call a contextualized actions. The context captures the conditions under which the action achieves the same effects as in the trace. Then, we associate a reward with each contextualized action. The reward associated with the contextualized action is designed so that agents will be rewarded for acting in a manner similar to their behavior in the team solution.

Using these contextualized actions and their rewards, together with the factored Dec-POMDP, we generate one single-agent problem for each agent. The dynamics of each single-agent problem is similar to that of the Dec-POMDP, except that some variables are projected away.

Finally, we process the single-agent policies and align them to try and ensure that actions are properly synchronized when they are executed in a decentralized manner. In the rest of this section we explain the steps that follow the generation of the team solution in more detail.

3.1 Producing the Traces

Having generated the team problem, $P_{\text{team}}$, we solve it and produce the traces, to capture the possible scenarios of the problem. We must specify three hyper-parameters: a pair of confidence parameters $\alpha$, $\beta$ and a precision parameter $\epsilon_{\text{team}}$. We generate an $\epsilon_{\text{team}}$-optimal solution to $P_{\text{team}}$ using an off-the-shelf POMDP solver, and then simulate that solution to produce $n_t$ traces. We want the empirical distribution of the initial states observed in the traces and the distribution over the team problem. We must specify three hyper-parameters: a pair of confidence parameters $\alpha$, $\beta$ and a precision parameter $\epsilon_{\text{team}}$.

To pick the number of traces we use a result on concentration bounds for multinomial distribution from (Agrawal 2019), since the initial belief state $b_0$ is a multinomial distribution. We denote by $T_0$ the sampled distribution, and by $k$ the number of initial states, namely the support set of $b_0$. Using the theorem, for every $n_t \geq \frac{\epsilon}{k\beta}$ we have:

$$Pr(KL(T_0||b_0) \geq \beta) \leq e^{-n_t \cdot \beta \left( \frac{e\beta n_t}{k-1} \right)^{k-1}}$$

Example 6. In our example, solving the team problem yields the following policy graph. Agent1 starts by pushing Box2 to the right, and then senses whether it had succeeded. It then moves left to assist Agent2 to push the heavy box, Box1, to the right, and again senses to verify its success.

Next, we use the policy graph to produce the traces. Different traces will differ by the number of push actions the agents perform until success. Table 1 shows two possible traces. Recall that the state is composed of 4 state variables: $(X_{A1}, X_{A2}, X_{B1}, X_{B2})$, where each variable can take values in $(L, R)$. The actions’ names will be denoted by the action name (M for move, P for push, CP for collaborative push and S for sense), followed by the direction for move and push actions $(L, R)$, and sub-scripted by the target box for sense and push actions $(B1, B2)$. We also denote the null-obs by $\phi$

<table>
<thead>
<tr>
<th>$X_{A1}$</th>
<th>$X_{A2}$</th>
<th>$X_{B1}$</th>
<th>$X_{B2}$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$\omega_1$</th>
<th>$\omega_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>R</td>
<td>R</td>
<td>L</td>
<td>PRB2</td>
<td>IDLE</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>L</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>$S_{B2}$</td>
<td>IDLE</td>
<td>$\phi$</td>
<td>no</td>
</tr>
<tr>
<td>L</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>MR</td>
<td>IDLE</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>CPLB1</td>
<td>CPLB1</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>L</td>
<td>R</td>
<td>$S_{B1}$</td>
<td>IDLE</td>
<td>no</td>
<td>$\phi$</td>
</tr>
<tr>
<td>L</td>
<td>R</td>
<td>R</td>
<td>L</td>
<td>PRB2</td>
<td>IDLE</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>L</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>$S_{B2}$</td>
<td>IDLE</td>
<td>$\phi$</td>
<td>no</td>
</tr>
<tr>
<td>L</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>MR</td>
<td>IDLE</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>CPLB1</td>
<td>CPLB1</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>$S_{B1}$</td>
<td>IDLE</td>
<td>yes</td>
<td>$\phi$</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>L</td>
<td>R</td>
<td>$S_{B1}$</td>
<td>IDLE</td>
<td>no</td>
<td>$\phi$</td>
</tr>
</tbody>
</table>

Table 1: An example of two traces

3.2 Extracting Contexted Actions

We seek a policy for each agent in which the agent’s public actions executions are identical to those that appear in the team plan. That is, the agent should execute the same public actions it executes in the team plan and in the same contexts. To generate such a policy, we define an appropriate reward function for each agent that encourages the agent to execute the public actions in the team plan in its own plan and in the same context.

The context of an action must capture the conditions under which the policy chooses the specific action to be executed. We can associate the context with a specific state, but this is too restrictive, as the state might contain various variables that are irrelevant to the action. It is preferable to define a less restrictive context that generalizes to all states where the action achieves the same effects.

Definition 1. The context of an action $a$ for agent $i$ is the set of values $\langle x_{j1}, \ldots, x_{jk} \rangle$ for the public variables and the private variables of agent $i$.

Definition 2. A contexted action (CA for short) is a pair $\langle c, a \rangle$, where $a$ is a public action and $c$ is the context of $a$, such that there exists a trace $t$ and an index $i$, where $t_i = \langle s, a, \omega \rangle$, and $c$ and $s$ assign identical values to the context variables of $a$ for agent $i$. 
We extract the CAs for agent $i$, denoted $CActions_i$, from the traces. For each trace $t$, we identify all the public actions in $t$. For each such public action $a$ of agent $i$ executed in a state $s$, we identify the context $c$ — the values that $s$ assigns to the public variables of the problem and private variables of agent $i$. In many cases, even though an action $a$ is executed in two different states $s_1, s_2$ in the traces, the context is identical, as we are interested only in the values of the state variables that are relevant to the execution. We focus on the public actions because the projected single agent problems are designed to plan execute these actions only in their appropriate context.

**Example 7.** Returning to our Box-Pushing example: we find the following public CAs of $Agent1$ in the traces:

- $(L,R,R,L), PR_{B2}$
- $(R,R,R,R), CPL_{B1}$

We construct $CActions_1$ for $Agent1$ by taking the values of the public variables of the problem, and the private variables of $Agent1$. The public variables are $X_{B1}, X_{B2}$, while the private variable of $Agent1$ is $X_{A1}$.

This results in the following CAs:

- $(X_{A1} = L, X_{B1} = R, X_{B2} = L), PR_{B2}$
- $(X_{A1} = R, X_{B1} = R, X_{B2} = R), CPL_{B1}$

We next describe how the single-agent problems are constructed, given the $CActions_i$ sets.

### 3.3 Single Agent Projection

Our next step is to define a factored POMDP $P_i$ for each agent $i$. $P_i$ is designed to incentivize the agents to execute the contexted actions of $i$ in the appropriate context. The actions of other agents are used to "simulate" some of the behaviors of the other agents — behaviors that eventually enable the agent to carry out its own actions. $P_i$ contains all state variables of the original problem. It also contains actions of agent $i$ and of other agents. Other agents’ actions are included to allow $i$ to simulate their behavior. More specifically, $P_i$ contains all and only the public actions that appeared in some trace of the team plan. In addition, $P_i$ contains all sensing actions of $i$, but not those of other agents.

For each private action $a$ of another agent, $P_i$ contains a deterministic version of $a$. Here, we use the assumption that each action has a known desired effect, and the deterministic version of $a$ always achieves this desired effect. This determination is done mainly to simplify $P_i$, allowing us to scale to larger problems. We can avoid this determination, at the cost of a more complicated $P_i$.

We now design the reward function of $P_i$. $R'_i$. The rewards incentivize the execution of the CAs in their appropriate context, while the penalties discourage the agent from applying them outside their context, so that its policy would emulate its behavior in the team plan.

When considering the single agent problems, we no longer want to reward the agents for achieving the team problem goals, but rather reward them for achieving their own parts of the team solution. Therefore, to make the new goals beneficial, we need their associated rewards to surpass the cost required to achieve them. To do so, we take a heuristic upper bounding approach, which manifests the following idea: In the single agent solutions, each CA will be preceded by a sequence of private actions. If the reward for applying that CA would surpass the total cost of its preceding actions, the agent will find it beneficial to apply. Furthermore, if satisfied for all CAs, the whole compound of single agent goals would become beneficial to achieve.

Let $MaxCost$ be the maximal negative reward received by a private action, that appeared in the traces. Recalling that we assume each action to have a single successful outcome, let $MinSP$ be the minimal success probability of all actions in the problem, and $MinCASP$ the minimal success probability of the actions appearing in the CAs. Let $MTL$ be the maximal trace length we produced, and $MPG$ be the Maximal Public Gap — the maximal number of private actions that precede a public action in the traces. We set $\epsilon > 0$ to some small positive value. Let $CA = (c, a)$ be a contexted-action and $Cost(CA)$ be the maximal negative reward received from $CA$ in the traces (and 0 if it was always positive). We compute $r_{CA}$, the reward given to $CA$:

$$\sum_{i=MTL-\text{MPG}}^{MTL-1}\frac{\gamma \cdot \text{MaxCost}}{\text{MinSP}} \cdot \gamma^{i-1} + \frac{\gamma \cdot \text{Cost(CA)} \cdot \gamma^{MTL-1}}{\text{MinCASP}} + \epsilon$$

The numerator is an upper bound on the expected discounted cost we would pay before applying CA as the last CA (the preceding sequence cost + the cost of the CA itself). The denominator amplifies that cost to be beneficial when scheduled as the last action in the policy.

As noted, we penalize the application of public actions in contexts other than those they appeared in in the team plan. This also eliminates potential positive reward cycles (Ng, Harada, and Russell 1999) that can cause an agent to endlessly achieve one of its sub-goals. The penalty is chosen to be $-\text{MinCASP} \cdot |CActions|$, as an upper bound on the sum of rewards that can be achieved from applying CAs. This ensures that public actions executed out of context cost more than applying all the CAs. There is no penalty for other agents’ CAs (i.e., contexted actions in $\bigcup_{j \neq i} CActions_j$). We want to allow the agent to simulate other agents’ CAs in order to plan the execution of its own actions at appropriate times, and also in order to act based on the uncertainty these actions can introduce.

Finally, we remove rewards related to public variables the agent can achieve because single-agent POMDP’s role is to imitate the team policy, not compute an alternative solution.

Given an action $a \in A_i$, a source state $s$ and a state $s' \neq s$ which differs from $s$ on at least one variable $X \in \text{eff}(a)$, we set $R'_i(s, a, s') = R(s, a, s') - \max(0, R(s, a, s') - R(s, a, s))$

**Example 8.** We now construct $Agent1$’s single-agent problem. We denote the CAs from the previous example with $ca_1, ca_2$, and their reward with $r_{ca_1}, r_{ca_2}$. We follow the projection stages one by one:

1. As the push action is the only public action in the problem, we remove all push actions except for the ones that are observed in the traces. For $Agent2$ we leave only $CPL_{B1}$ and for $Agent1$ we leave $CPL_{B1}$ and $PR_{B2}$. 

Notice that we keep Agent2’s $CPL_{B1}$ action in Agent1’s problem, as we might need to simulate it.

2. We remove the sensing action of Agent2, as well as its observation variable.

3. We don’t have any non-deterministic private actions, so no actions are turned to deterministic.

4. We add a penalty of $-2 \cdot \max(r_{ca1}, r_{ca2})$ to the remaining public actions, applied in any context except for the CA’s contexts.

5. We add the rewards $r_{ca1}, r_{ca2}$ to $ca1$ and $ca2$ respectively.

6. The rewards for pushing the boxes to the target cells are set to 0 - we reward the agent only for doing its public action in context.

### 3.4 Policy Adjustment and Alignment

We run the planner on each of the single agent projections that we generate, constructing a set of single agent policy graphs for the projections. We now adapt the policies to apply to the joint problem.

First, the projection of agent $i$ contains private actions of other agents that must be removed. We traverse through the policy graph and replace every action of another agent by its child. As we do not allow sensing actions of other agents, there is always a single child to actions of other agents.

We now align the policies to increase the chance 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 an action $a_2$ of agent 2 should be executed before $a_2$. Also, collaborative actions should be executed at the same time by all participating agents.

Given stochastic action effects, we cannot guarantee that the policies will be well correlated. It is desirable that we will maximize the probability of synchronization, but currently, we only offer a heuristic approach that empirically improves synchronization probability.

For each public action $a$ in the team plan we select a simple path from the root to $a$. The identifier of the action is the sequence of public actions along the simple path. Public actions in individual agent policy graphs that share the same identifier are assumed to be identical in all graphs.

For a public action $a$ that has the same identifier in all agent graphs, let $l$ be the length of the longest simple path to the action in all 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-ops. Then, we move to the next action, and so forth. After the alignment we remove from each agent’s aligned policy graph all the public actions of other agents.

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. To do so, after executing the action, both agents perform a sensing action that senses the effect of that collaborative action. In executions, the agents may be unsynchronized, applying the collaborative and sensing actions in an alternating manner, where one agent performs the collaborative action while the other performs the sensing action, causing them to enter a livelock. To handle that, 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. This way, a livelock can never occur.

**Example 9.** Figures 1(a) and 1(b) show Agent1’s policy graph before and after the alignment and adjustments procedure. Since Agent2’s job is only to collaboratively push Box1 with Agent1, there are no action simulations of Agent2 in Agent1’s policy, hence no alignment is required. The only modification that occurs is the insertion of the live-lock handling.

### 4 Empirical Evaluation

We provide experimental results focusing on longer planning horizons and scaling up with respect to current Dec-POMDP solvers. The experiments were conducted on a variation of the popular cooperative box pushing problem. We compare our algorithm, FDMAP, with two Dec-POMDP solvers, GMAA-ICE (Oliehoek et al. 2013) and JESP (Nair et al. 2003), using MADP-tools. (Oliehoek et al. 2017). We evaluate FDMAP, GMAA-ICE and JESP on a Linux machine with 4 cores and 8GB of memory.

#### 4.1 Collaborative Box-Pushing

In the cooperative box pushing domain, agents on a grid can move and push boxes in four principle directions, or perform a no-op. Light boxes can be pushed by a single agent, while heavy boxes can only be pushed by a collaborative push of two agents. All actions except for the push actions are deterministic.

Each grid cell can contain any number of agents and boxes, and each agent can sense for a box at its present location. Initially, each box can appear in either the target cell (and hence, need not be moved) or the lower right cell, with equal probability. The goal of the agents is to move the boxes to a target cell, located at the upper left corner of the grid.

Each action (except for no-op) has a cost — 10 for moving, 1 for sensing (encouraging sensing rather than blindly pushing), 30 for pushing, and 20 for a collaborative push (per agent). The reward for moving a box to its target position is 500. In addition, there’s a penalty of 10000 for pushing a box out of the target cell to avoid abuse. In configurations with heavy boxes we double the reward and penalty. A domain instance of $m$ cells with $n$ agents, $l$ light boxes, and $h$ heavy boxes, has $m^{n+l+h}$ states, $(5 \cdot (l+1) + 4h \cdot (n-1))^{n}$ actions and $3^{n}$ observations.

Evidently, the box pushing domain calls for longer horizon policies, rather than good local reactive policies, and requires careful coordination to ensure that collaborative push actions are performed simultaneously by the agents.

#### 4.2 Settings

We compared FDMAP with GMAA-ICE and DP-JESP. GMAA-ICE and DP-JESP require an horizon specification,
while FDMAP computes a policy for an unbounded horizon. Therefore, we specify the maximal reached planning horizon for them under the column -H-, and for FDMAP we specify the average number of steps until reaching the goal state, under the column -Avg-. Discount factor is set to $\gamma = 0.99$.

For GMAA-ICE and DP-JESP we report the computed policy value. For FDMAP we measured the average discounted accumulated reward of 1000 simulations.

We provide results on several configurations of Collaborative Box-Pushing. Each box starts at either the top-left or bottom-lower corner with equal probabilities. The problem name convention is composed of 5 marks (each mark specified in brackets), $BP - [w][h][n][l][h]$. The marks stand for Width, Height, Number of agents, Number of light boxes, Number of heavy boxes. For 1 dimensional grids, DP-JESP and GMAA-ICE were given a minimalistic version of the problem that does not include unnecessary actions — push and move for the up and down directions — to decrease the domain size. We specify the agents’ initial locations (denoted by $I$), as well as the domain size, alongside the configuration name.

All planners were given a total of 3600 seconds to solve each $\langle$configuration, horizon$\rangle$ pair. In addition, we also limited the solution of each single-agent problem withint FDMAP to 900 seconds. For the hardest problem configuration (BP-33221), we also present results for larger time limit, as 900 seconds were not sufficient for the agents to solve their $P_i$.

The time shown for DP-JESP and GMAA-ICE is based only on its log from MADP-Toolbox, and does not include the problem loading, which is in many cases non negligible. FDMAP time does not include writing the SARSOP policies and graphs to disk, as they are highly dependent on hardware quality and can effectively remain in memory throughout the whole process.

In both GMAA-ICE and DP-JESP, configurations BP-32302, BP-32303 and BP-33221 could not be solved for any horizon. Therefore we provide comparisons only for the first four configurations (Table 2), while the harder configurations are shown in Table 3. We also provide a more sensitive analysis of the horizon in table 5.

In DP-JESP, $\times$ marks a timeout. In GMAA-ICE we mark two different timeout options: FF refers to failure of finding a full policy for the required horizon, where FH refers to an earlier stage timeout when computing the heuristic function.

### 4.3 Results

The main comparison is presented in Table 2. We can see that FDMAP manages to produce policies with higher quality, as these policies require horizons beyond those that the other planners can handle. We also see that FDMAP’s planning time is significantly smaller. This is due to the fact that FDMAP does not plan directly on the decentralized model, but rather solves multiple POMDP models, which are known to require much less computational effort (Bernstein, Zilberstein, and Immelman 2013).

For the largest problems, shown in Table 3, FDMAP still manages to produce good quality policies, yet with signifi-
Table 2: The results for configurations BP-31211, BP-22202 and BP-22203. FDMAP outperforms DP-JESP and GMAA-ICE with respect to both running time and policy value. The running time is improved significantly. Results for DP-JESP and GMAA-ICE are for maximal horizon reached, specified under the -H- column. In FDMAP we present the average number of steps until reaching the goal state when running the policy for an unbounded number of steps, specified under the -Avg- column.

<table>
<thead>
<tr>
<th>Problem</th>
<th>MaxSteps</th>
<th>AvgSteps</th>
<th>Time</th>
<th>Value</th>
<th>%Wins</th>
</tr>
</thead>
<tbody>
<tr>
<td>BP-32302</td>
<td>40</td>
<td>14</td>
<td>92.70</td>
<td>243.91</td>
<td>100</td>
</tr>
<tr>
<td>BP-32303</td>
<td>43</td>
<td>21</td>
<td>1799.94</td>
<td>353.82</td>
<td>98</td>
</tr>
<tr>
<td>BP-33221</td>
<td>124</td>
<td>37</td>
<td>2962.29</td>
<td>8.64</td>
<td>95</td>
</tr>
<tr>
<td>BPPEN-3221</td>
<td>105</td>
<td>35</td>
<td>5379.32</td>
<td>345.56</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 3: Results for the largest scale problems, which only FDMAP managed to solve. Running times are rapidly increasing while reaching the scales limit of the underlying POMDP solver. The policies are still very robust, and reach the goal state in most cases (%Wins). MaxSteps and AvgSteps specify the maximal and average number of steps made until reaching a goal state throughout the runs.

Table 4: Results for a variation of Collaborative Box-Pushing, in which we penalize an agent for pushing a light box blindly.

<table>
<thead>
<tr>
<th>Problem</th>
<th>MaxSteps</th>
<th>AvgSteps</th>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BP-32302</td>
<td>40</td>
<td>14</td>
<td>92.70</td>
<td>243.91</td>
</tr>
<tr>
<td>BP-32303</td>
<td>43</td>
<td>21</td>
<td>1799.94</td>
<td>353.82</td>
</tr>
<tr>
<td>BP-33221</td>
<td>124</td>
<td>37</td>
<td>2962.29</td>
<td>8.64</td>
</tr>
<tr>
<td>BPPEN-3221</td>
<td>105</td>
<td>35</td>
<td>5379.32</td>
<td>345.56</td>
</tr>
</tbody>
</table>

Table 5: Results for BP-21210. FDMAP outputs a reasonable value compared to GMAA-ICE, which optimizes the small scaled problem. The last row presents results for the maximal horizon reached on DP-JESP and GMAA-ICE, and average simulations steps until reaching the goal state for FDMAP when run for unbounded number of steps. These are specified in parentheses next to the policy value.

Table 5 shows that FDMAP manages to produce reasonable results compared to the other solvers even when dealing with extremely small domains in which optimal solvers such as GMAA-ICE can excel. To observe the difference between the policies FDMAP produces to the ones GMAA-ICE does, we present another variation of the domain, where we add a penalty of 101 to light boxes push actions that occur with no box in the cell. The penalty is chosen to be slightly higher than the reward for pushing a box to the target cell, times the fail probability of the push action. Table 4 presents the results (configuration name prefixed with BPPEN). We can see that the the reward difference on maximal results compared to Table 2 are much lower for FDMAP, indicating that FDMAP’s policies in the non-penalty configurations exploit the horizon and avoid blindly pushing, leading to higher-quality results. If we would handle domains in which reward can be only achieved in large horizons, a case that was mentioned in contexts of reward shaping (Brys et al. 2014; Laud and DeJong 2003), we expect FDMAP’s ability to scale to very large horizons while managing to reward agents for their public goals, to become crucial.

5 Conclusion And Future Research

We presented FDMAP — an algorithm for solving factorized Dec-POMDPs. FDMAP begins by solving a centralized POMDP, which we call the team POMDP, obtaining a team plan. Then, FDMAP creates agent specific POMDPs whose solutions encourage agents to complete their role in the team plan. The agent plans are then aligned for synchronization between the agents. We experiment with box pushing examples that require collaboration, showing that we can scale to much larger problems than current Dec-POMDP solvers, while computing a reasonable policy.

There are two direction in which we deem FDMAP can be improved, in both scalability and solution quality. The use of online planners instead of SARSOP as the underlying POMDP solver, can greatly improve the scale of solvable
problems. The changes in terms of algorithm’s structure are minor, as we merely need to be able to produce the single agent policy graphs using an online solver. In terms of solution quality, we aim at using more principled methods of reward shaping, that come from the field of reinforcement learning in forms of multi-objectivization (Brys et al. 2014). Our goal would be to convert the concept of contexted actions into objectives of each agent, while preserving optimality with respect to the decentralized problem.

Acknowledgements
This work was supported by ISF Grants 1651/19, by the Israel Ministry of Science and Technology Grant 54178, and by the Lynn and William Frankel Center for Computer Science.

References


2014. Pomdpx file format (version 1.0).


