A Factored Approach to Deterministic Contingent Multi-Agent Planning

Shashank Shekhar
Department of Computer Science
Ben Gurion University
shekhar@cs.bgu.ac.il

Ronen I. Brafman
Department of Computer Science
Ben Gurion University
brafman@cs.bgu.ac.il

Guy Shani
Information and Software Systems Eng.
Ben Gurion University
shanigu@bgu.ac.il

Abstract

Collaborative Multi-Agent Planning (MAP) under uncertainty with partial observability is a notoriously difficult problem. Such MAP problems are often modeled as Dec-POMDPs, or its qualitative variant, QDec-POMDP, which is essentially a MAP version of contingent planning. The QDec-POMDP model was introduced with the hope that its simpler, non-probabilistic structure will allow for better scalability. Indeed, at least with deterministic actions, the recent IMAP algorithm scales much better than comparable Dec-POMDP algorithms (Bazinin and Shani 2018). In this work we suggest a new approach to solving Deterministic QDec-POMDPs based on problem factoring. First, we find a solution to a MAP problem where the results of any observation is available to all agents. This is essentially a single-agent planning problem for the entire team. Then, we project the solution tree into sub-trees, one per agent, and let each agent transform its projected tree into a legal local tree. If all agents succeed, we combine the trees into a valid joint-plan. Otherwise, we continue to explore the space of team solutions. This approach is sound, complete, and as our empirical evaluation demonstrates, scales much better than the IMAP algorithm.

Introduction

In many real-world problems agents collaborate to achieve joint goals. For example, disaster response teams typically consist of multiple agents that have multiple tasks to perform, some of which require the cooperation of multiple agents. In such domains, agents typically have partial information, as they can sense their immediate surroundings only. As agents are often located in different positions and may possess different sensing abilities, their runtime information states differ. Sometimes, this can be overcome using communication, but communication infrastructure can be damaged, or communication may be costly and should be reasoned about explicitly.

In this setting it is common to plan for all agents jointly using a central engine. The resulting policy, however, is executed by the agents in a decentralized manner, and agent communication is performed only through explicit actions.

Decentralized POMDPs (Dec-POMDPs) offer a rich model for capturing such multi-agent problems (Bernstein et al. 2002; Oliehoek and Amato 2016), but Dec-POMDPs solvers have difficulty handling larger problems. Qualitative Dec-POMDP (QDec-POMDP) were introduced as an alternative model, replacing the quantitative probability distributions over possible states with qualitative sets of states (Brafman, Shani, and Zilberstein 2013). At least for deterministic problems, i.e., where partial observability plays the key role, QDec-POMDP algorithms scale much better than Dec-POMDP algorithms. The original translation-based algorithm of (Brafman, Shani, and Zilberstein 2013) scaled somewhat better than contemporary Dec-POMDP algorithms, and the recent IMAP algorithm (Bazinin and Shani 2018) scales much better than current Dec-POMDP algorithms (Bazinin and Shani 2018). In this paper, too, we focus on deterministic QDec-POMDP problems, and demonstrate even better scalability than previous methods.

The policy for a QDec-POMDP can be represented as a joint policy tree (or graph), where nodes are labeled by joint actions of all agents, and edges are labeled by joint observations. A solution to a QDec-POMDP is a policy tree, all of whose leaf nodes correspond to goal states.

A single agent (a.k.a. local) policy tree contains only that agent's actions and observations. In the search for such trees, the branching factor can be much smaller than that of the joint-policy space. Thus, one would expect that generating multiple local trees would be easier than a single joint policy, provided the overhead of ensuring that these local policies are properly coordinated is not too large. Indeed, the approach used by the IMAP algorithm (Bazinin and Shani 2018) is to solve multiple single-agent problems until all trees are coordinated. (At least abstractly, this is reminiscent of the iterated best-response method used by the JESP algorithm for Dec-POMDPs (Nair et al. 2003).) Roughly speaking, in IMAP, each agent solves a single-agent problem in a domain that contains all of its actions, as well as actions belonging to other agents, as well as all original goal conditions. The next agent solves a similar problem, but where its goals are to perform all its actions that appear in the previous agents' solutions. If one agent is unable to solve its problem, backtracking occurs.

In this paper we present an alternative approach for solving QDec-POMDPs in which single-agent problems are solved repeatedly, too. This approach is motivated by the ideas behind factored algorithms for classical planning in

which factoring was obtained by defining the problem as a MAP problem (Brafman and Domshlak 2008). Not only does this approach show superior empirical performance, it is also cleaner algorithmically and hence comes with a simple completeness guarantee, has greater potential for concurrency – allowing true distributed computation by a group of agents, and is likely to be extendable to offer privacy preserving properties, as well.

The approach works as follows: First, we solve a MAP problem in which we assume that communication is free and immediate. Hence, all agents "see" each others' observations. This is a *single-agent* planning problem in which the actions available are the union of all agents' actions, and the observations are the union of all agent's observation actions. Thus, while this problem is not as simple as that of generating a policy for a single agent, because we have a larger action and observation space, it is not a MA planning problem: we need to track only a single belief state, and we do not need to reason about multiple concurrent belief states and to coordinate between agents with different states of information.[1] We refer to this as the *team planning problem*.

From the policy tree obtained by solving the team problem, we extract for each agent a sub-tree that contains only the actions of that agent that impact other agents. These could be collaborative actions (i.e., actions that require joint concurrent execution by multiple agents, such as lifting a table or pushing a heavy box), or actions that supply preconditions to other agents' actions. We refer to this as the agent's projection of the policy tree.

Agent φ_i's projection is not likely to be executable by it for two possible reasons. First, it may require φ_i to perform action a only when p holds, where p was observed before in the original team policy by another agent φ_j. In the team problem, φ_i learns the results of such observations immediately and for free, but in the real domain, φ_i must somehow obtain this information. Second, the projected solution removes all φ_i's action that are not needed by other agents. Some of these removed actions may have supplied some precondition to one of the remaining actions. Thus, the next step in the algorithm is to let each agent turn its projected policy into an executable policy. To do this, each agent solves a local planning problem in which its goal is to perform all the actions in its projection under the same conditions. Thus, its solution would be a policy that the agent can execute (provided the other agents execute their actions in the team policy).

Of course, the single-agent problems may not be all solvable, in which case we must backtrack and seek a new team solution. But if they are all solvable, then we can get a legal joint policy for the original problem by taking the solutions of each projected problem and aligning its actions properly so that collaborative actions are executed at the same time, and preconditions are supplied before their consuming actions. Thus, if the underlying single-agent contingent planner is able to generate all solutions (e.g., by using AO^* as the underlying search algorithm), then our method is complete.

We implemented and tested this approach using the single-agent CPOR planner, an off-the-shelf offline contingent planner (Komarnitsky and Shani 2016) that generates a solution in the form of a policy graph. We compare it with IMAP on the two domains described in that paper, and on a new disaster support domain. Our factored planning algorithm scales better both as the number of objects in the domain increase and as the number of agents increases. In addition, it typically generated smaller, and more balanced policy trees.

Model Definition

We start with the basic definition of a flat-space QDec-POMDP, followed by a factored definition motivated by contingent planning model definitions (Bonet and Geffner 2014).

Definition 0.1. A qualitative decentralized partially observable Markov decision process (QDec-POMDP) is a tuple $\mathcal{Q} = \langle I, S, b_0, \{A_i | i \in I\}, \delta, \{\Omega_i\}, O, G \rangle$ where

- I is a finite set of agents indexed $1, ..., m$. We often refer to the i^{th} agent as φ_i.
- S is a finite set of states.
- $b_0 \subset S$ is the set of states initially possible.
- A_i is a finite set of actions available to agent φ_i, and $\vec{A} = \otimes_{i \in I} A_i$ is the set of joint actions, where $\vec{a} = a_1, ..., a_m$ denotes a particular joint action.
- $\delta : S \times \vec{A} \rightarrow 2^S$ is a non-deterministic Markovian transition function. $\delta(s, \vec{a})$ denotes the set of states the can be reached when taking joint action $\vec{a}$ in state s.
- Ω_i is a finite set of observations available to agent φ_i and $\vec{\Omega} = \otimes_{i \in I} \Omega_i$ is the set of joint observation, where $\vec{o} = o_1, ..., o_m$ denotes a particular joint observation.
- $\omega : \vec{A} \times S \rightarrow 2^{\vec{\Omega}}$ is a *non-deterministic* observation function. $\omega(\vec{a}, s)$ denotes the set of possible joint observations $\vec{o}$ given that joint action $\vec{a}$ was taken and led to outcome state s. Here $s \in S$, $\vec{a} \in \vec{A}$, $\vec{o} \in \vec{\Omega}$.
- $G \subset S$ is a set of goal states.

We do not assume here a finite horizon T, limiting the maximal number of actions in each execution. We focus, however, on deterministic outcomes and deterministic observations. In such cases a successful solution is acyclic, hence, there is no need to bound the number of steps. Extension to domains with non-deterministic outcomes with a bounded horizon is simple, but extensions to infinite horizon and non-deterministic outcomes is beyond the scope of this paper. We assume a shared initial belief, like most Dec-POMDP models, which is most natural for an off-line centralized algorithm (again, like most Dec-POMDP algorithms).

We will work with a factored representation of a QDec-POMDP, specified using the following components: $\langle I, P, \{A_i | i \in I\}, Pre, \mathit{Eff}, Obs, b_0, G' \rangle$ where I is a set of agents, P is a set of primitive propositions, $\vec{A}$ is a vector of individual action sets, Pre is the precondition function, Obs

[1] For flat models, single-agent contingent planning is in PSPACE (Littman, Goldsmith, and Mundhenk 1998) and MA contingent planning is NEXP-TIME hard (Bernstein et al. 2002).

is an observation function, *Eff* is the effects function, b_0 is the initial state formula, and G is a set (conjunction) of goal propositions.

We now explain the QDec-POMDP induced by such a factored QDec-POMDP specification. First, its state space, S, consists of all truth assignments to P, and each state can be viewed as a set of literals. Its initial states and goals are all states that satisfy the initial state formula and the goal conjunction, respectively.

Its transition function δ is defined using Pre and *Eff* as follows: The precondition function Pre maps each individual action $a_i \in A_i$ to its set of preconditions, i.e., a set of literals that must hold whenever agent φ executes a_i. Preconditions are local, i.e., defined over a_i rather than $\vec{a}$, because each agent must ensure that the relevant preconditions hold prior to executing its part of the joint action. We extend Pre to be defined over joint actions $\{\vec{a} = \langle a_1, .., a_m\rangle : a_i \in A_i\}$ (where $m = |I|$): $Pre(\langle a_1, .., a_m\rangle) = \cup_i Pre(a_i)$.

Brafman *et. al.* [2013] define an effects function *Eff* mapping joint actions into a set of pairs (c, e) of conditional effects, where c is a conjunction of literals and e is a single literal, such that if c holds before the execution of the action e holds after its execution. Thus, effects are a function of the *joint* action rather than of the local actions, as can be expected, due to possible interactions between local actions. However, in line with (Bazinin and Shani 2018; Shekhar and Brafman 2018), here we give more structure to joint actions.

We assume that single-agent actions executed concurrently do not interact, unless specified explicitly. Such interactions are then modeled by collaborative actions. Collaborative actions have the same form as single-agent actions, except that they have multiple agent parameters. Thus, an agent may have a single-agent *move* action, as well as participate in a collaborative, two-agent action, *joint-lift*, for lifting a table. One can think of *joint-lift* as two concurrent single-agent *lift* actions (e.g., as modeled in (Shekhar and Brafman 2018)). If a collaborative action such as *joint-lift* exists, and single-agent *lift* exist, too, then it is forbidden for the planner to schedule two separate single-agent *lift* actions at the same time. If it wishes to perform the two *lift* actions concurrently, it must use the *joint-lift* action. For a deeper discussion of the issue of defining joint actions, see (Shekhar and Brafman 2018). We remark here that, when one does not allow concurrent actions that delete one another's preconditions or object capacity constraints (Crosby, Jonsson, and Rovatsos 2014), then one can consider only joint actions that consist of a single (possibly collaborative) action at each step with all other agents performing no-ops, greatly simplifying the process. Later, the plan can be made more compact in post-processing, e.g., using the technique of (Crosby, Jonsson, and Rovatsos 2014).

For every joint action $\vec{a}$ and agent φ, $Obs(\vec{a}, i) = \{p_1, \ldots, p_k\}$, where $p_1, ..., p_k$ are the propositions whose value agent φ observes after the joint execution of $\vec{a}$. The observation is private, i.e., each agent may observe different aspects of the world. We assume that the observed value is correct and corresponds to the post-action variable value. In our domains, we will separate actions into observation and non-observation actions. The former do not affect the world state, and the latter have an empty set of observations. Every action can be separated into a non-observation and an observation action by adding suitable propositions forcing the two to appear consecutively in every plan.

While QDec-POMDPs allow for non-deterministic action effects as well as non-deterministic observations, we focus in this paper only on deterministic effects and observations, and leave discussion of an extension of our methods to non-determinism to future research.

Policy Trees

We can represent the local plan of an agent φ using a *policy tree* τ_i, which is a tree with branching factor $\leq |\Omega|$. Each node of the tree is labeled by an action, and edges that follow an observation are labeled by an observation. To execute the plan, each agent performs the action at the root of the tree and then uses the subtree labeled with the observation it obtained for future action selection. If τ_i is a policy tree for agent φ and o_i is a possible observation for agent φ, then $\tau_{i_{o_i}}$ denotes the subtree that is rooted by the child of the root of τ_i that is reached via a branch labeled by o_i.

Let $\vec{\tau} = \langle \tau_1, \tau_2, \cdots, \tau_m\rangle$ be a vector of policy trees, also called a *joint policy*. We denote the joint action at the root of $\vec{\tau}$ by $\vec{a}_{\vec{\tau}}$, and for an observation vector $\vec{o} = o_1, \ldots, o_m$, containing each agent's observation, we define $\vec{\tau}_{\vec{o}} = \langle \tau_{1_{o_1}}, \ldots, \tau_{m_{o_m}}\rangle$.

Because (unlike Dec-POMDPs) actions may have preconditions, a joint policy tree is executable only if the preconditions of each action hold prior to its execution. To check this, we must maintain the sets of states possible at each point in time during the execution of the joint policy. This is usually referred to as the *belief state*. Notice that this is the belief state of the entire system, not of a single agent. Online, each agent will have less information, because it cannot distinguish between all branches of the joint-policy. However, here we are taking the point of view of the off-line planner. To follow policy $\vec{\tau}$, we first consider the action $\vec{a}_{\vec{\tau}}$ given the current belief state b. It must be the case that $b \models pre(a_\tau)$. In that case, we say that $\vec{a}_{\vec{\tau}}$ is *executable* in b. After the agents execute a_τ and observe $\vec{o}$, their new belief state is $tr(b, \vec{o}, \vec{a}_\tau) = \{a_\tau(s) | s \in b, a_\tau(s) \models \vec{o}\}$.

We say that joint policy $\vec{\tau}$ is *executable* given initial belief b if (1) $\vec{a}_{\vec{\tau}}$ is executable in $\vec{b}$; (2) if a_{τ_i} is a part of a collaborative action and j is another agent participating in that collaborative action, then a_{τ_j} contains j's part of that action; (3) For every possible joint observation $\vec{o}$, $\vec{\tau}_{\vec{o}}$ is executable given $tr(b, \vec{o}, \vec{a}_{\vec{\tau}})$.

A joint policy is called a *solution* if it is executable, and for all leaf nodes in the tree $\bigcap_i b_i \models G$, i.e., the set of possible states given the joint local beliefs of the agents satisfy the goal. Note that unlike Dec-POMDPs, for QDec-POMDPs there is no obvious notion of optimal policy, or optimization criterion, although one could strive to find trees with smaller depth, or trees that minimize the maximal branch cost.

Example 1. *We now illustrate the factored QDec-POMDP model using a simple box pushing domain (Figure 1). In this example there is a one dimensional grid of size 3, with cells*

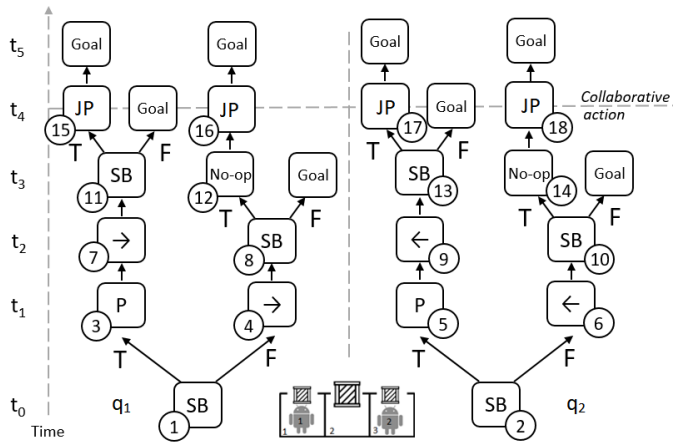

Figure 1: Illustration of Example 1 showing the box pushing domain with 2 agents and a possible set of local plan trees that produce a solution. Possible agent actions are sensing a box at the current agent location (denoted SB), moving (denoted by arrows), pushing a light box up alone (denoted P), jointly pushing a heavy box (denoted JP), and no-op.

marked 1-3, and two agents, starting in cells 1 and 3. In each cell there may be a box, which needs to be pushed upwards. The left and right boxes are light, and a single agent may push them alone. The middle box is heavy, and requires that the two agents push it together.

We can hence define $I = \{1, 2\}$ *and* $P = \{AgentAt_{i,pos}, BoxAt_{j,pos}, Heavy_j\}$ *where* $pos \in \{1, 2, 3\}$ *is a possible position in the grid,* $i \in \{1, 2\}$ *is the agent index, and* $j \in \{1, 2, 3\}$ *is a box index. In the initial state each box may or may not be in its corresponding cell —* $b_0 = AgentAt_{1,1} \wedge AgentAt_{2,3} \wedge (BoxAt_{j,j} \vee \neg BoxAt_{j,j})$ *for* $j = 1, 2, 3$*. There are therefore 8 possible initial states.*

The allowed actions for the agents are to move left and right, to push a light box up, or jointly push a heavy box up with the assistance of the other agent. There are no preconditions for moving left and right, i.e. $Pre(Left) = Pre(Right) = \phi$*. For agent* φ *to push up a light box* j*, agent* φ *must be in the same place as the box. That is,* $Pre(PushUp_{i,j}) = \{AgentAt'_{i,j} \neg Heavy_j, BoxAt_j\}$*. For the collaborative joint push action the precondition is* $Pre(JointPush_j) = \{AgentAt_{1,j}, AgentAt_{2,j}, Heavy_j, BoxAt_j\}$*.*

The moving actions transition the agent from one position to the other, and are independent of the effects of other agent actions, e.g.,
$Right_i = \{(AgentAt_{i,1}, \neg AgentAt_{i,1} \wedge AgentAt_{i,2}), (AgentAt_{i,2}, \neg AgentAt_{i,2} \wedge AgentAt_{i,3})\}$*.*
The only joint effect is for the JointPush action — $Eff(PushUp_{1,2}, a_2)$ *where* a_2 *is some other action, are identical to the independent effects of action* a_2*,*
while $Eff(PushUp_{1,2}, PushUp_{2,2}) = \{(\phi, \neg BoxAt_{2,2})\}$*, that is, if and only if the two agents push the heavy box jointly, it (unconditionally) gets moved out of the grid.*

We define sensing actions for boxes — $SenseBox_{i,j}$*, with precondition* $Pre(SenseBox_{i,j}) = AgentAt_{i,j}$*, no effects, and* $Obs(SenseBox_{i,j}) = BoxAt_{j,j}$*. The goal is to move all boxes out of the grid, i.e.,* $\bigwedge_j \neg BoxAt_{j,j}$*.*

In the following we will use the term *projected sub-tree (or graph)* to denote a (policy) tree that is obtained from the original tree by removing some nodes. Whenever a node is removed, its parent becomes the parent of the children of the removed node. If we have a policy tree containing actions of two types A and B, and we want to look at the projected sub-tree containing A nodes only, we can imagine iteratively removing all B nodes, until none are left.

A Factored Approach to Solving QDec-POMDPs

We now present a very general scheme for factored planning in QDec-POMDPs. In fact, in principle, with suitable modifications and a suitable single-agent contingent planner, this approach works for non-deterministic domains, as well.

The high-level structure of the algorithm is described below in Algorithm 1.

Algorithm 1: Factored Planning for QDEC-POMDP

```
1:  Input: QDec-POMDP ⟨I, P, A⃗, Pre, Eff, Obs, b0, G⟩
2:  Set P_team = ⟨P, ∪_{i=1}^m A_i, b0, G⟩)
3:  while unexplored solutions to P_team exist do
4:     τ_team = Contingent-Solve-Next(P_team)
5:     for all agent φ_i do
6:        τ_i = Project(τ_team, φ_i)
7:        P_i = Generate-Contingent(τ_i)
8:        τ'_i = Contingent-Solve(P_i)
9:        if unsolvable P_i then
10:          GOTO 4
11:       end if
12:    end for
13:    (τ''_1, ..., τ''_m) = Align(τ'_1, ..., τ'_m)
14:    return (τ''_1, ..., τ''_m)
15: end while
16: Fail
```

First, we generate the single-agent team problem. This is simply obtained by taking the original MAP problem, treating all actions as if they are executed by a single agent and all observations are observed by a single, "combined" agent. This results in a team solution tree denoted τ_{team}. Next, τ_{team} is projected to each agent, obtaining τ_i for $i = 1, \ldots m$. Now, each agent solves a planning problem whose goal is to generate an executable local policy that contains τ_i as a projected sub-tree. That is, each action in τ_i is executed in this local policy under the same conditions and in the same order. If all problems are solvable then we align the actions in their solution and return a solution. If one of the agents cannot solve its local problem, we generate a new team solution and repeat the process. If no new team solution remains, we fail.

The Team Problem Generating the team problem is easy. We take the original QDec-POMDP and simply treat all

agents as objects under the control of some super-agent. This super-agent also receives all the observations. This team problem is now a single-agent contingent planning domain.

The Single-Agent Problems Once we have a solution τ_{team} to the team problem, we generate one projection τ_i of it for every agent. The projection is obtained by removing from the tree all non-observation actions except those executed by agent φ_i. Among the actions of agent φ_i, we leave only actions that impact other agents directly. An action impacts another agent directly if either (1) it is a collaborative action; (2) it supplies a precondition to an action of another agent; or (3) it achieves a goal proposition. In factored planning such actions are often referred to as *public* actions, while the remaining actions are called *private* (Brafman and Domshlak 2008). Similarly, a proposition that appears in the description of multiple agents is called *public*, and a proposition that appears in the description of only a single agent's actions is called *private*. For example, ignoring collaborative actions for the moment, in the Box Pushing example, the *push* actions of different agents are public, assuming multiple agents can push the same box. The *move* actions, however are private. Similarly the location of a box, which can be influenced by multiple agents is public, while the location of an agent, that can be influenced only by its *move* actions, is private.

Collaborative actions are a bit more subtle. A *joint-push* action is performed by multiple-agents. We treat it as a public action. However, when considering whether a proposition is private or not, we consider only its part of the action. Thus, *joint-push* requires both agents to be located in the same position, making an agent's location appear public. However, since we will ensure that joint-actions are respected by all participating agents, the other agents need not be concerned with the preconditions of these actions that are otherwise private to other agents. Thus, one agent need not know or care about the location of the other agent – this is the latter agent's problem – as long as that agent executed the collaborative action at the same time.

Thus, each projection τ_i is a tree containing observation actions, possibly by other agents, and the public actions of agent φ_i. Furthermore, if a is an action in τ_i, we remove from it any public precondition, that is, one supplied by another agent in τ_{team}.

Next, we must ensure that all observations are necessary. Consider, for example, a case in which τ_i includes sensing the value of p, but the agent acts identically whether p is *true* or *false*. The reason the observation for p exists in the original team solution τ_{team} is probably because some other agent needs to differentiate between these two cases. If we leave this distinction in place, we risk losing completeness if our agent is unable to observe p.

To remove redundant observations, we apply standard graph algorithms: Moving bottom-up, whenever the two sub-trees below an observation node are identical, we remove the observation and retain just one of the sub-trees. When comparing sub-trees, two sensing actions by different agents that sense the same proposition are treated as identical, and only the "single-agent" element of the collaborative actions is considered – i.e., we do not distinguish between two lift-table actions in which the other agents collaborating with φ_i are different, because the action φ_i executes in this collaboration is the same. An example of a team solution, its projection, and the compacted projection is given in Figure 2.

The projected tree is typically not executable by the agent. It contains observation actions that are not its own, and some actions are not supplied with their preconditions by previous actions. Our goal is to extend this tree, by replacing the observation actions of other agents by the agent's observation actions, when relevant, and adding private actions that supply missing preconditions. Note that only private actions are added – if an agent adds a public action that requires that another agent will supply it with a precondition, or one that changes the value of a public proposition, this might impact the other agents, either requiring them to modify their plan, or destroying a precondition they need.

Under the assumptions that all other agents execute their public actions in their projection, this tree is executable because all actions have the preconditions supplied either by other agents or by the agent. The resulting policy tree, τ_i^{sol}, should have the property that, when projected to contain only observations and public actions of φ_i, it is identical to τ_i, with the exception that observations of other agents are replaced by those of φ_i.

Thus, the next step is to take the compact policy tree obtained (which is also denoted τ_i) and generate a single-agent contingent planning problem. The goal of solving this problem is to generate an executable policy tree that contains all the actions in the projected tree, where these actions are executed under the same conditions and in the same order. This planning domain is generated as follows:

1. The set of actions contains all ground actions appearing in τ_i and all private action of φ_i;
2. Preconditions that other agents achieved in τ_{team} are removed;
3. Each action in a leaf node has the added effect $done$; in case of a branch in which execution terminates after an observation, a dummy action that achieves $done$ is appended.
4. Each non-leaf action a has a special added effect p_a;
5. Each non-root action a has an added precondition $p_{a'}$, where a' is the parent of a in the tree;
6. The first action in a branch following an observation has the appropriate value of the observation as an added precondition. If there are multiple consecutive observations without intermediate actions, then the value of all of them in this branch have to be in its preconditions.
7. Initially, all added propositions are *false*;
8. The goal of the planning problem is $done$.

Example 2. *Consider the projected tree in Figure 2(C). For this tree, we generate a planning domain for agent 2 with the following actions:*

- *The original description of Agent 2's private actions – in this domain these are the various movements of the agent.*

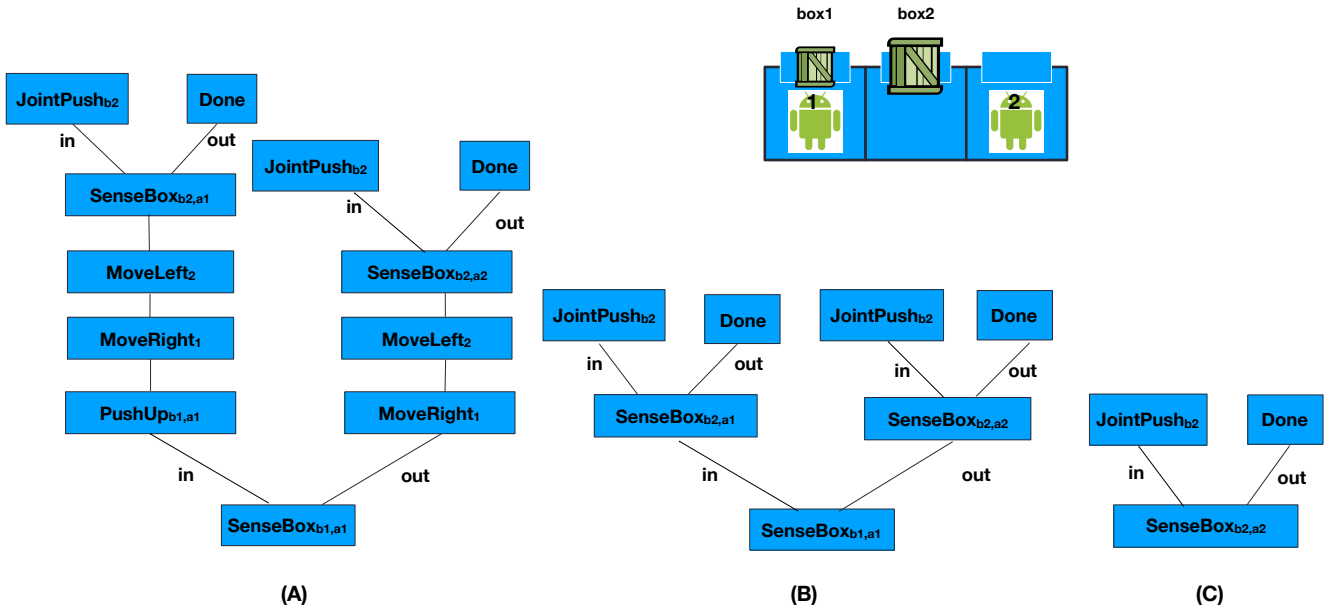

Figure 2: (A) A team solution plan for a problem with two agents, a light box and a heavy box that need to be outside the grid in the goal state. (B) Its projection to Agent 2. Notice that observations include those of Agent 1, too. (C) Compacted projection – no sensing is required by Agent 1.

- *The action* $SenseBox_{2,2}$ *with an added effect:* $observed_{box2,a2}$.
- *The action Dummy-Out with preconditions* $SensedBox_{2,2}$ *and* $\neg BoxAt_2$ *(the observed value). Its only effect is* $done$.
- *The action* $JointPush_2$. *Its preconditions contain Agent 2's private preconditions, i.e.,* $AgentAt_{2,2}$ *and the observation value for this branch:* $BoxAt_2$. *Its effects are the original effects and* $done$.

The added propositions: $SensedBox_{2,2}$ *and* $done$ *are initially* false.

Alignment If all projected problems are solvable, we still need to align them to ensure that the joint-policy is executable. This is done by concurrently going over all trees level by level and ensuring that all actions are executable and that all collaborative actions are executed at the same step. If an action is not executable at the current step, or if only one part of a collaborative action is scheduled to a time-step, a *no-op* is added to the relevant branch so that the execution of the action is delayed to the next step. We show that this is possible in the next section.

Soundness and Completeness

Theorem 1. *The factored planning algorithm is sound if the underlying single-agent contingent planner is sound.*

Proof. First, we observe that the joint policy contains all public actions that appear in the team solution. If some action is missing then we cannot achieve $done$ in the branches that contain it. Because we add branch conditions as preconditions, only execution of branch actions in sequence can make it *true*.

Next, we observe that the solution plan is executable. All preconditions supplied originally by other agents are supplied by them by the observation above. All other preconditions have to be supplied by the agent itself to obtain a valid plan, and soundness of the single-agent planner implies its validity. Of course, other agents' actions that supply an agent's preconditions must be scheduled before and cannot be destroyed. Let k be the maximal number of actions inserted between any two public actions of any branch in any agent's plan. Consider the original team plan, but now with k no-ops inserted between any two consecutive actions. This plan remains a valid team plan. Now, we have enough time steps to insert all the additional private plans in between these actions without impacting the relative order of public actions. Note that this also ensures that collaborative actions are executed at the same time. Since no new public actions can be added when solving the projected problems, one agent's solution cannot introduce actions that might delete a precondition. Hence the solution is executable.

Finally, since all goal achieving actions are public, then by the above they will be executed and the goal will be achieved in all branches, hence this policy is a solution. □

Theorem 2. *The factored planning algorithm is complete provided the single-agent contingent planning algorithm can exhaustively generate all possible team policies.*

Proof. Suppose that the multi-agent planning algorithm has a solution. It is also a solution to the single-agent algorithm, and hence it will be generated by it at some point. Its projection will contain all the public actions and observations that the agent executes in the solution. Since there is a solution, the local projections are solvable, and their solution

contains all needed additional private actions. As explained in the soundness proof, there is a simple, less efficient variant of the alignment algorithm that is guaranteed to succeed if there is a team solution. □

We observe that we do not need to generate the multi-agent solution as a team solution to generate a valid solution. It is enough that we generate a team solution that contains all the public actions of the solution of the MAP problem and that these actions will appear under the conditions in which they appear in the MAP solution. As discussed earlier, we can obtain an exhaustive single-agent solver either by running an algorithm such as AO^* or by using a complete solver and augmenting the domain with constraints that rule-out past solutions.

Note, however, that if the single-agent algorithm generates only plans with non-redundant actions, the algorithm is, in general, incomplete. We discuss this and related issues in the next section.

Compromises for Efficiency

A top level that exhaustively searches the space of team policies is unlikely to work in practice. To make the algorithm more efficient, we have made a few compromises. We explain them and their implications for completeness.

Backtracking Algorithm 1 requires that solutions be generated one after another. In principle, this is easy to do with a systematic tree-generation algorithm such as AO^*. However, currently CPOR (Komarnitsky and Shani 2016) does not support AO^*-based search, and we suspect that a AO^* planner for contingent planning is likely to be inefficient due to the lack of good heuristics for such problems. An alternative is to augment the planning domain to reflect learned nogoods so that previously generated solutions are no longer solutions for the new domain. As this only causes us to prune inappropriate team solutions, this does not affect the algorithm's completeness.

The fundamental problem is to change the domain of a single-agent contingent planner so that a previous solution will not be generated again. Consider team solution τ which was not extendable to a joint plan. We want to make sure that no future solution will be τ, or τ with some of its branches extended with new actions.

This requires that at least one branch of τ will not appear as a branch prefix in a following solution. Given a *specific* branch, we can ensure that it is not part of future solutions by adding suitable preconditions and effects to the actions in this branch, so that if they are executed in sequence, we reach a dead-end. We can then force the planner to select a specific branch of τ by requiring the first action in the plan to be one of a set of actions, each of which essentially selects one branch. However, this approach is not scalable, as this modification is required following each backtrack.

For efficiency, we actually add a weaker, but simpler constraints (albeit, ones that are still not very scalable). The main source of failure on the projected problem is the need to branch on some condition p that is not observable by the agent, followed by sub-trees that are different depending on the reason for failure. Hence, our first step is to traverse the projected sub-tree and find such branch points.

Suppose we have an asymmetric sub-tree rooted in an observation of p. Suppose that a is an action that appears on one branch following the observation, but not on the other branch. We would like to add a constraint that forces a to appear on both branches, or not to appear at all. We add a special action *commit-p* that can only appear before the observation of p. This action has a special effect that is add as a precondition to a. Thus, if p was observed, a cannot be executed unless, earlier, we executed *commit-p*. This action commits to performing a on all branches following the observation: it has an effect that negates a goal proposition, and this effect can be negated by a only.

Signalling

Consider a problem in which agent φ_1 can observe p and agent φ_2 can observe q, but not p, and that the solution requires that agent φ_2 will act differently depending on p's value. In general, the problem is unsolvable. However, suppose that φ_1 can control the value of q. It can then signal to φ_2 the value of p by manipulating q. In theory, if we generate every possibly team policy, we can generate such a policy as well. There is a caveat, though. The team solution will branch on p, φ_1 will align the value of p with q and then it will branch on q. For our algorithm to remove branching on p in φ_2's projection, the two sub-trees that correspond to the two possible values of p must be identical. A decent single-agent contingent planner will not insert a redundant observation – and since observing q adds no value in this case, it is redundant. Furthermore, even if it does, the branches given $p \wedge \neg q$ and $\neg p \wedge q$ will be left empty, as they never occur.

One ad-hoc way of addressing this is using signalling procedures to replace observations. That is, if the projected problem for φ_2 requires sensing p, we can try to replace it in the plan by some signalling sub-routine/macro followed by an observation of the signal. This undermines the separation between the team solution and local solutions, since signalling involves two agents. However, it is likely to be practical. Another option is to try to keep track of the belief states of agents during the team planning. To some extent, this can be done syntactically, and was done in earlier work on compiling conformant and contingent planning into classical planning (Palacios and Geffner 2009). However, this approach does not scale too well, and this issue remains a key challenge for our method.

Empirical Evaluation

In this section we provide an empirical analysis that shows that our approach scales much better than IMAP. The IMAP paper considers two domains: Box-Pushing (BP) and Rovers, and we add a novel domain called Heavy-Structural-Damage (HSD) domain. We describe the domains below.

Box-Pushing In this domain there are boxes spread in a grid like structure. Each box must be moved to its destination at the edge of the grid, basically the end of the column it appears in. Each box is either at some location or at the goal location. An agent needs to be in the same grid cell where

a box exists, to sense/observe the box, and to push it. Boxes are either heavy or light, and to push a heavy box two agents are required to perform a collaborative push action while a single agent can push a light box. In addition, an agent can move to adjacent locations (four primary directions). In this domain we have uncertainty about the locations of the boxes.

Heavy-Structural-Damage This domain captures a scenario where in a grid like locality, due to an earthquake, several buildings have been collapsed. Debris is scattered all around and there is the possibility that underneath some pillar/beam of a collapsed building there is a victim. Agents need to search the victims and rescue them. An agent can *sense* a patient. If the patient is underneath a pillar or a roof beam, the patient will be rescued to a safe location. If the patient is an adult a joint rescue operation will be performed by more than one agent. Agents can move to connected locations in the maze. The goal is to rescue all the victims.

Rovers This is an adaptation of multi-agent Rovers domain, in which multiple rovers (agents) must collect together measurements of soil and rock. A rover navigates between two waypoints on a map and to collect a measurement the rover must be present at some waypoint that is unknown to the rover initially. A rover at a waypoint can attempt a measurement, but whether the rover would measure things successfully is actually based on if the waypoint is appropriate for that measurement. Images of rocks and samples of soils can be collected by a single rover. For rock samples it requires two rovers working jointly. After taking measurements, the rovers must broadcast them back to the ground station.

Experiments Both planners were run on a Windows 10, 64 bit machine with *i*7 processor, 2.8GHz CPU, and 16Gb RAM. Both our factored approach and IMAP are implemented in C#. For IMAP, we used the code by Bazinin and Shani (2018).

The results are shown in Table 1. We describe the running time, and size of the resulting plan tree which is measured in terms of the number of branches and the height. The number of branches is also indicative of the number of sensing actions performed, as branching occurs following an observation. We depict only the maximum values of width and height of individual plan trees obtained for all the agents. Table 1 shows that the factored planning approach scales better than IMAP. Specifically, the increase in the number of objects had minor impact on running time, as opposed to IMAP. Even more markedly, increasing the number of agents had a much smaller influence on the running time of the factored planning algorithm. Thus, except for the smallest problem, the factored planning algorithm is faster than IMAP and scales to much larger problems. And, except for only a few problems, the policy trees generated by the factored planning algorithm are smaller. In fact, upon examining the policies generated, we observe that the factored planning approach often generates more balanced trees for different agents, while IMAP generates trees with significantly different sizes. Finally, in the HSD domain, we can also see that increasing the number of agents does not have a major impact on run-time of the factored planning approach.

Domain	Ins (#agt)	Size	Factored Approach *vs* IMAP					
			Max Width		*Max Height*		*Time (sec)*	
Box-Pushing	P01 (3)	12	8	8	10	11	1.7	8.7
	P02 (3)	15	16	16	26	15	4.6	18.8
	P03 (4)	16	4	8	5	17	1.4	44.6
	P04 (5)	19	4	32	4	15	1.6	97.5
	P05 (5)	21	38	-	20	-	13.6	-
	P06 (6)	25	4	128	9	31	2.7	130.2
	P07 (9)	36	64	-	24	-	25.3	-
	P08 (10)	37	90	-	29	-	41.1	-
	P09 (12)	46	64	-	23	-	33.9	-
	P10 (12)	46	128	-	24	-	43.8	-
	P11 (12)	48	128	-	26	-	60.2	-
HSD	P01 (3)	14	8	8	11	9	1.2	7.1
	P02 (3)	14	6	8	15	12	2.4	8.6
	P03 (4)	20	8	20	9	14	5.1	70.7
	P04 (6)	32	4	64	7	29	3.2	208.9
	P05 (7)	36	4	128	7	33	3.8	402.8
	P06 (7)	36	112	-	31	-	35.6	-
	P07 (8)	40	110	-	29	-	53.8	-
	P08 (8)	42	148	-	36	-	81.5	-
	P09 (9)	47	128	-	31	-	88.3	-
	P10 (9)	47	127	-	34	-	129.5	-
Rovers	P01 (1)	12	2	2	9	9	0.5	0.4
	P02 (2)	14	2	2	8	8	0.7	0.8
	P03 (1)	12	4	4	15	15	0.8	1.5
	P04 (2)	17	11	12	21	43	3.1	20.8
	P05 (2)	17	12	-	24	-	3.5	-
	P06 (2)	17	27	27	43	52	7.2	267.6
	P07 (2)	28	35	-	35	-	8.5	-
	P08 (2)	28	31	-	37	-	7.6	-

Table 1: Performance comparison of our factored approach and IMAP approach. *Ins* is instance number with the number of acting agents in the brackets. *Size* denotes the number of objects considered in each problem. *Maximum Width* and *Maximum Height* respectively show the values of maximum number of branches and maximum height of all individual solution trees obtained for the agents. Time is in seconds.

In most of the tested domains, the agents are homogenous – they have the same actions. In these domains, backtracking is not needed. The instances P02 and P05 in the Box-Pushing domain, and the instances P02 and P03 in the HSD domain contain non-homogenous agents. This causes backtracking to occur in all of these problems because when an agent is asked to act differently following a sensing action's results, and the agent cannot perform this sensing action, then agent roles must be changed. We can see that in Box-Pushing P05, the need to backtrack caused the planner to take more time than a slightly larger problem that does not require backtracking (P06). Similarly, in HSD, P03 takes more time than the slightly larger P04. Nevertheless, the running times were still quite reasonable and in all these instances, the factored planner did much better than IMAP.

Recall that IMAP was compared to Dec-POMDP algorithms on the BP domain, and scaled much better. Thus, one can reasonably conclude that, for the special class of deterministic Dec-POMDPs where the uncertainty is only w.r.t. the initial state, our factored approach is able to handle much larger instances than Dec-POMDP algorithms.

Summary

We described a factored approach to solving multi-agent contingent planning problems. The problems were modelled using the qualitative QDec-POMDP model, and the solution approach works by taking a team solution – i.e., one in which all agents have immediate access to every other agent's observations – and fixing it so that it becomes executable by all agents. In our experimental evaluation we compared the factored planning algorithm with IMAP, a recent algorithm that, at least on deterministic problems, scales to much larger domains than current algorithms for Dec-POMDPs. The factored planning algorithm is almost always faster, and often much faster than IMAP, scales better with increased agent numbers, and typically generates smaller policy trees.

Problem factoring is a key element in MAP in general (e.g., (Ephrati and Rosenschein 1997)) and in many algorithms for Dec-POMDPs (e.g., (Oliehoek et al. 2008; Witwicki and Durfee 2010; Kumar, Zilberstein, and Toussaint 2011; Oliehoek, Witwicki, and Kaelbling 2012). It is particularly natural to factor the problem among the different agents, yielding a distributed algorithm. In particular, we believe it is natural to study algorithms in which the information exchanged by agents centers on their *commitments* to other agents. This is essentially what our algorithm does – each agent tries to plan to fulfil its commitments in the team plan. While commitments bring to mind the idea of *influences*, studied in the Dec-POMDP literature (Witwicki and Durfee 2010; Oliehoek, Witwicki, and Kaelbling 2012), they are different. Influences are used to separate the belief state of one agent from all other information – they capture the variables that directly influence the agent. Distributions representing variables external to the agent can be marginalized to these influences, reducing the states space that an agent needs to consider, e.g., when computing best response. Commitments can be viewed as an analogous concept at the level of the policy – they refer to the actions that an agent performs in order to facilitate the actions of another agent.

The factored planning approach has a number of advantages: it is conceptually simple, and hence easy to analyse theoretically; the second stage of the algorithm generates independent sub-problems for all the agents, and can be parallelized; and since each agent solves a local planning problem, it is likely that a privacy preserving variant can be formulated. On the other hand, a method that tries to generate a MA policy directly, instead of first committing to a team plan, may lead to better solutions. Moreover, efficient backtracking and signalling are still quite challenging, and it was not tested on non-deterministic domains. So while it scales significantly better than Dec-POMDP solvers on deterministic problems, more work is needed to extend its applicability to more complex situations.

Acknowledgments We thank the reviewers for their useful comments. This work was supported by ISF Grant 1210/18, the Israel Ministry of Science and Technology Grant 54178, the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Center of Ben-Gurion University of the Negev, and the Lynn and William Frankel Center for Computer Science

References

Bazinin, S., and Shani, G. 2018. Iterative planning for deterministic QDec-POMDPs. In *GCAI 2018*, 15–28.

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27:819–840.

Bonet, B., and Geffner, H. 2014. Belief tracking for planning with sensing: Width, complexity and approximations. *J. Artif. Intell. Res.* 50:923–970.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS*, 28–35.

Brafman, R. I.; Shani, G.; and Zilberstein, S. 2013. Qualitative planning under partial observability in multi-agent domains. In *AAAI'13*.

Crosby, M.; Jonsson, A.; and Rovatsos, M. 2014. A single-agent approach to multiagent planning. In *ECAI*, 237–242.

Ephrati, E., and Rosenschein, J. S. 1997. A heuristic technique for multi-agent planning. *Ann. Math. Artif. Intell.* 20(1-4):13–67.

Komarnitsky, R., and Shani, G. 2016. Computing contingent plans using online replanning. In *AAAI'13*, 3159–3165.

Kumar, A.; Zilberstein, S.; and Toussaint, M. 2011. Scalable multiagent planning using probabilistic inference. In *IJCAI 2011*, 2140–2146.

Littman, M. L.; Goldsmith, J.; and Mundhenk, M. 1998. The computational complexity of probabilistic planning. *Journal of AI Research* 9:1–36.

Nair, R.; Tambe, M.; Yokoo, M.; Pynadath, D.; and Marsella, S. 2003. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *IJCAI 2003*.

Oliehoek, F. A., and Amato, C. 2016. *A Concise Introduction to Decentralized POMDPs*. Springer Briefs in Intelligent Systems. Springer 2016.

Oliehoek, F. A.; Spaan, M. T. J.; Whiteson, S.; and Vlassis, N. A. 2008. Exploiting locality of interaction in factored Dec-POMDPs. In *AAMAS 2008*, 517–524.

Oliehoek, F. A.; Witwicki, S. J.; and Kaelbling, L. P. 2012. Influence-based abstraction for multiagent systems. In *AAAI*.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *J. Artif. Intell. Res. (JAIR)* 35:623–675.

Shekhar, S., and Brafman, R. I. 2018. Representing and planning with interacting actions and privacy. In *ICAPS 2018*, 232–240.

Witwicki, S. J., and Durfee, E. H. 2010. From policies to influences: a framework for nonlocal abstraction in transition-dependent Dec-POMDP agents. In *AAMAS'10*, 1397–1398.