EPiC Series in Computing

Volume 55, 2018, Pages 15–28

GCAI-2018. 4th Global Conference on Artificial Intelligence

Iterative Planning for Deterministic QDec-POMDPs

Sagi Bazinin and Guy Shani

Information and Software Systems Engineering, Ben Gurion University

Abstract

QDec-POMDPs are a qualitative alternative to stochastic Dec-POMDPs for goal-oriented planning in cooperative partially observable multi-agent environments. Although QDec-POMDPs share the same worst case complexity as Dec-POMDPs, previous research has shown an ability to scale up to larger domains while producing high quality plan trees. A key difficulty in distributed execution is the need to construct a joint plan tree branching on the combinations of observations of all agents. In this work, we suggest an iterative algorithm, IMAP, that plans for one agent at a time, taking into considerations collaboration constraints about action execution of previous agents, and generating new constraints for the next agents. We explain how these constraints are generated and handled, and a backtracking mechanism for changing constraints that cannot be met. We provide experimental results on multi-agent planning domains, showing our methods to scale to much larger problems with several collaborating agents and huge state spaces.

1 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 [1], but Dec-POMDPs solvers have difficulty to scale up beyond small toy problems. Qualitative Dec-POMDP (QDec-POMDP) were offered as an alternative model, replacing the quantitative probability distributions over possible states with qualitative sets of states [6].

Although QDec-POMDPs share the same worst case complexity class as Dec-POMDPs, Brafman et al. have shown that a translation-based approach into contingent planning managed to scale to model sizes that could not be solved by Dec-POMDP algorithms. 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 the possible joint observations following those actions. In a solution tree all the leaves correspond to goal states. The policy tree hence has a huge branching factor, limiting the ability

to scale up to larger problems. One can extract single agent local policy trees from the joint policy tree, where each local tree has an exponentially smaller branching factor. In this paper we focus on deterministic QDec-POMDPs, where one can find solutions with a finite depth.

In many problems interactions between cooperating agents are limited to a number of key points. Each agent may be able to achieve a set of tasks that require no cooperation, while assisting other agents only in several collaborative tasks. This is illustrated in the box pushing domain (Figure 2), where light boxes can be pushed into place by a single agent, but a heavy box can only be pushed by two agents together.

In such cases, it may be useful to plan for each agent independently, creating a single agent plan tree, branching only on the observations of the specific agent. We suggest an iterative approach, which we call IMAP (iterative multi agent planning) where the central planning engine plans for one agent at a time. IMAP creates a local policy tree for each agent, instead of a joint policy tree for all agents.

When planning for a single agent IMAP assumes that other agents will be available to assist in required collaborations. These assumptions generate a set of conditional constraints on the behavior of other agents, that must be considered when planning for these agents. When a constraint cannot be satisfied, we backtrack to the agent that required the constraint.

In addition, agents attempt to perform tasks at the lowest cost, notifying all other agents of the cost for completing subgoals. Agents that manage to complete some tasks more cheaply, inform backward to previous agents, that replan again ignoring these tasks. Thus, our approach also contains a task allocation component that assigns tasks to agents to reduce the overall cost for completing all tasks.

For solving single agent problems, we compile the multi-agent QDec-POMDP into a single agent contingent planning problem. We use an off-the-shelf offline contingent planner [9] to generate a plan graph, and then extract the conditional constraints from that plan graph. Our method is sound, but incomplete, due to the greedy nature of our iterative process. Still, it scales up to very large QDec-POMDPs. We provide an empirical study, focusing on scaling up analysis, showing how our approach scales to very large domains, with multiple agents, many order of magnitudes beyond domains solvable by previous approaches.

2 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 [2, 5].

Definition 2.1. A qualitative decentralized partially observable Markov decision process (QDec-POMDP) is a tuple $\mathcal{Q} = \langle I, S, b_0, \{A_i\}, \delta, \{\Omega_i\}, O, G\rangle$ where

- I is a finite set of agents indexed $1, ..., m$.
- 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 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.
- $\delta : S \times \vec{A} \to 2^S$ is a non-deterministic Markovian transition function. $\delta(s, \vec{a})$ denotes the set of possible outcome states after taking joint action $\vec{a}$ in state s.
- Ω_i is a finite set of observations available to agent 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.
- $\omega : \vec{A} \times S \to 2^{\vec{\Omega}}$ 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, and there is hence 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.

Definition 2.2. A factored QDec-POMDP is a tuple $\langle I, P, \vec{A}, Pre, \mathit{Eff}, Obs, b_0, G\rangle$ where I is a set of agents, P is a set of propositions, $\vec{A}$ is a vector of individual action sets, Pre is the precondition function, and Obs is an observation function. Eff is the effects function. b_0 is the set of initially possible states, and G is a set (conjunction) of goal propositions. The state space S consists of all truth assignments to P, and each state can be viewed as a set of literals.

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 i 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. [6] 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.

We slightly refine these definitions to explicitly support independent and collaborative actions. For each local action a_i of agent i we define a set of local conditional effects $\mathit{eff}_l(a_i) = \{(c, e)\}$. In addition, for each subset of local actions of agents, $\{a_{i_1}, ..., a_{i_k}\}$ of agents $i_1, ..., i_k$ we define another set of collaborative conditional effects $\mathit{eff}_c(\{a_{i_1}, ..., a_{i_k}\}) = \{(c, e)\}$. When this set is empty, we say that the local actions are independent, when the set is not empty, we say that the subset of local actions are collaborative. We further require that collaborative subsets will be minimal, in that for two subsets A^1, A^2 such that $A^1 \subset A^2$, $\mathit{eff}_c(A^1) \neq \mathit{eff}_c(A^2)$. While the effects of collaborative actions are shared, the preconditions are not. Specifically, when applying a collaborative action each agent must only ensure that its own preconditions hold.

For every joint action $\vec{a}$ and agent i, $Obs(\vec{a}, i) = \{p_1, \ldots, p_k\}$, where $p_1, ..., p_k$ are the propositions whose value agent i 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.

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. Additionally, our model assumes a shared initial belief state, as most Dec-POMDP models. The challenging case where the initial belief states differs between agents is important, as it corresponds to the situation in on-line planning, but it is left for future research.

2.1 Policy Trees

We can represent the local plan of an agent i using a *policy tree* τ_i, which is a tree with branching factor $|\Omega|$. Each node of the tree is labeled with an action and each branch is labeled with an observation. To execute the plan, each agent performs the action at the root of the tree and then uses the subtree labeled

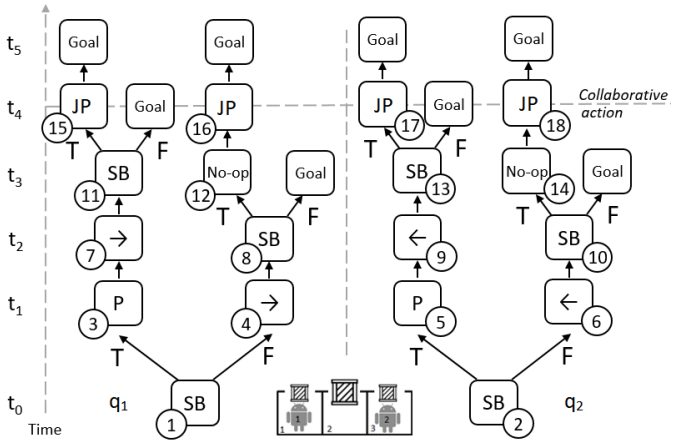


Figure 1: Illustration of Example 1 showing the box pushing domain with 2 agents and a possible set of local plan trees that produce a solution. Possible agent actions are sensing a box at the current agent location (denoted SB), moving (denoted by arrows), pushing a light box up alone (denoted P), jointly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

with the observation it obtains for future action selection. If τ_i is a policy tree for agent i and o_i is a possible observation for agent i, then $\tau_{i_{o_i}}$ denotes the subtree that corresponds to the branch labeled by o_i.

An alternative to local trees is a global joint policy tree, where nodes are labeled by joint actions and edges are labeled by joint observations. We argue that constructing global trees directly is difficult, due to the need to consider all possible observation combinations together, and suggest an iterative construction of local policy trees as a more scalable approach.

Let $\vec{\tau} = \langle \tau_1, \tau_2, \cdots, \tau_m \rangle$ be a vector of policy trees. We denote the joint action at the root of $\vec{\tau}$ by $\vec{a}_{\vec{\tau}}$, and for an observation vector $\vec{o} = o_1, \ldots, o_m$, we define $\vec{\tau}_{\vec{o}} = \langle \tau_{1_{o_1}}, \ldots \tau_{m_{o_m}} \rangle$. When executing a policy tree each agent i maintains its own local belief b_i — the set of possible states given the observations that i has observed during the execution. Initially, all agents set $b_i = b_0$, but following the different observations it is often the case that $b_i \neq b_j$ for two agents i and j. $tr(\vec{b}, \vec{o}, \vec{a})$ denotes the set of local beliefs after executing the joint action $\vec{a}$ and observing $\vec{o}$ starting from the local beliefs $\vec{b}$. To execute $\vec{\tau}$ we first consider the action $\vec{a}_{\vec{\tau}}$ in the context of the local beliefs. That is, each agent i must validate that $b_i \models pre(a_{\tau_i})$. If for some agent j, $b_j \not\models pre(a_{\tau_j})$ then the execution is not valid. Each agent i then executes a_{τ_i}, observes o_i, transitioning to one of the subtrees τ_i' of τ_i with a new local belief b_i'. We say that $\vec{\tau}$ is a valid set of policy trees if every possible execution for $\vec{\tau}$ starting from b_0 is valid. If the precondition of $\vec{a}_{\vec{\tau}}$ are met at the initial local beliefs $\langle b_0, ..., b_0 \rangle$, and for every possible joint observation $\vec{o}$, executing $\vec{\tau}_{\vec{o}}$ in the new local belief $tr(\vec{b}, \vec{a}_{\vec{\tau}}, \vec{o})$ is valid.

A local policy tree τ_i is valid if there exists a valid vector $\vec{\tau}$ of local policy trees containing τ_i. For example, a local policy tree τ_i is not valid if for some t, for two different branches of τ_i of length t, a collaborative action with some agent j appears in one branch but not in the other, and agent j cannot distinguish between the two branches. That is, some differentiating observations cannot be observed by j.

A set of policy trees $\vec{\tau}$ is called a *joint policy* if executing the policy trees starting from the initial belief b_0 results in a valid execution. A joint policy is called a *solution* if for all leaves 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.

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 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* i *to push up a light box* j*, agent* i *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.,* $\bigwedge_j \neg BoxAt_{j,j}$*.*

3 Iterative Construction of Policy Trees

We now describe our main contribution — a method to construct satisfying policy trees iteratively, which we call IMAP, for iterative multi agent planning. The first agent constructs an independent policy tree using only its own independent actions, and collaborative actions that it participates in, assuming that the other agents required to execute the collaborative actions will be available to assist. This single agent problem is solved by a contingent planner [9], returning a policy tree.

After the single agent policy tree is computed, the agent manipulates the tree to be a valid local policy tree in a QDec-POMDP. Then, the agent extracts constraints for other agents, encapsulating the assistance requirement of the computed policy.

We now create a new single agent problem for the next agent, containing the computed constraints. The next agent attempts to solve this constrained problem. If the agent succeeds, it again extracts constraints and passes them on. If the single agent contingent planning problem cannot be solved, however, the agent reports the failure back, with additional information, and the first agent must replan.

The process terminates when all agents agree on a set of local policy trees that achieve the goal. Then, we run a soundness check. The following subsections describe the various parts of this algorithm. For ease of exposition, we describe our methods below assuming time, or unit costs. Our methods are directly applicable to the case of varying costs.

A high level description of IMAP is presented in Algorithm 1.

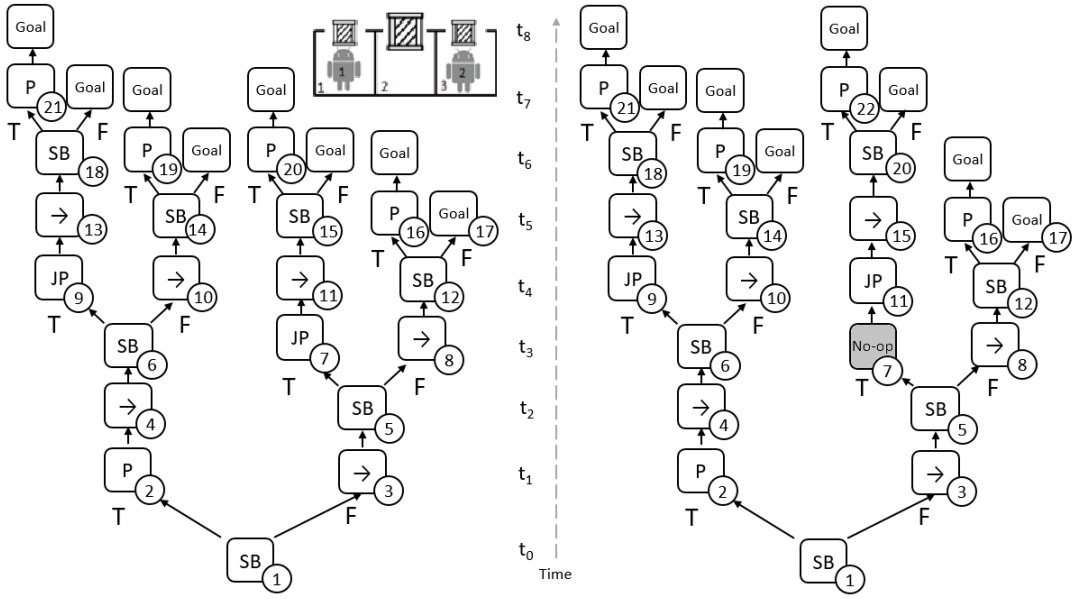


Figure 2: Policy tree (left) and an adjusted tree (right) for the compiled contingent problem of agent 1 in the simple box pushing domains. A *no-op* action (grayed) was inserted to level the JointPush action to be executed at time t_4

3.1 Compilation to a Single Agent Problem

We now describe the creation of a single agent problem for the first agent 1. The planning problems for the next agents will be based on this compilation, adding constraints which we later describe. Given a factored QDec-POMDP $\langle I, P, \vec{A}, Pre, \mathit{Eff}, Obs, b_0, G\rangle$ we create for agent i a single agent problem $\langle P_i, A_i^+, Pre_i, \mathit{Eff}_i, Obs_i, b_0, G\rangle$. The observations Obs_i, the initial belief b_0 and the set of goals G remain as in the multi agent problem.

A_i^+ is the set of single agent i, containing all the independent actions in A_i, as well as all collaborative actions that i participates in.
That is, for each minimal subset $\{a_{j_1}, ..., a_{j_k}\}$ such that $\mathit{Eff}_c(\{a_{j_1}, ..., a_{j_k}\}) \neq \emptyset$ and there exists l such that $a_{j_l} \in A_i$, we add a single agent action $a_{\{j_1,...,j_k\}}$. The created action has the same preconditions as a_{j_l} — the component of agent i in the joint action. This represents an assumption of agent i that the other agents that participate in the collaborative action will fulfill the needed precondition for the collaborative action to apply.

In addition, we identify a set P_i^- of non-constant propositions that none of the independent or collaborative actions of agent i can achieve, yet appear in a precondition of an action $a \in A_i^+$ or in the goal G. These are propositions which may be required for achieving the goal, yet cannot be produced by agent i. We remove all these propositions from the problem description. This represents an assumption of agent i that other agents will produce these propositions when needed.

We can now run a contingent planner on the compiled problem and obtain a single agent policy tree τ.

Algorithm 1: Iterative Planning for QDEC-POMDP

Input: QDec-POMDP $\langle I, P, \vec{A}, Pre, Eff, Obs, b_0, G\rangle$
$A_c \leftarrow \emptyset$ (collaborative action constraints)
$i \leftarrow 1$ (current agent)
for each goal literal g, $G^T(g) \leftarrow \infty$
while $i < n$ **do**
  $\tau_i \leftarrow$ solve $\langle P_i, A_i^+, Pre_i, Eff_i, Obs_i, b_0, G\rangle$
  if τ_i *is valid* **then**
    Align and order collaborative actions in τ_i
    $G_i^T \leftarrow$ goal achievement times in τ_i
    if $\exists g \in G$ *s.t.* $G_i^T(g) < G^T(g)$ **then**
      $j \leftarrow$ the earliest agent that achieved g s.t. $G_i^T(g) < G^T(g)$
      $G^T(g) \leftarrow G_i^T(g)$
      Undo all constraints by agents j .. i
      $i \leftarrow j$
    for *Collaborative action* a_c *in* τ_i *at time* t **do**
      Add constraint on a_c at time t to A_c
    $i \leftarrow i + 1$
  else
    for *constraint* c *in* A_c *by increasing time* **do**
      $\tau \leftarrow$ Solve $\langle P_i, A_i^+, Pre_i, Eff_i, Obs_i, b_0, c\rangle$
      if τ *is not valid* **then**
        $c_f \leftarrow c$
        $t \leftarrow$ earliest time that c_f could be achieved
    $j \leftarrow$ the agent that introduced c_f
    notify j that c_f can be achieved by time t
    Undo all constraints by agents j .. i
    $i \leftarrow j$

3.2 Adjustments to the Policy Tree

It may not be possible to construct a joint policy using τ, as we must ensure that all collaborating agents execute a collaborative action together. For example, consider the single agent policy tree for agent 1 in Figure 2. This tree is a solution for the compiled problem, assuming that agent 2 would assist in joint-push actions when needed. However, we can observe that in the left branch, the collaborative joint push action is at time t_4, while in the right branch the joint push is at time t_3. To be able to assist at different times, agent 2 must know whether agent 1 is at the right or left branch.

To handle this, we force all the instances of a collaborative action to occur at the same time in all branches, by adding *no-op* actions (Figure 2 right), creating a *leveled* policy tree.

Leveling the collaborative actions may be difficult when we have multiple collaborative actions in a branch. Given two collaborative actions a_1, a_2, if a_1 precedes a_2 in all branches, then we can level the execution time as before, by first leveling the tree for a_1, then leveling the tree for a_2. However, when a_1 precedes a_2 in one branch, and a_2 precedes a_1 in another branch, we cannot level both together.

There can be several ways to create a valid leveled policy tree. First, we can replan forcing the planner to decide on one ordering of all collaborative actions. There can be cases where this will make the problem unsolvable. We can also condition the difference on an observed variable that all collaborating agents can observe. We currently take the first approach, and fail if the planner cannot order all collaborative actions in a consistent order of execution in all branches.

3.3 Extracting Constraints for Other Agents

Following the adjustments to the tree τ, resulting in tree τ_i for agent i, we now extract a set of constraints on the policy tree of other agents. We can extract two types of constraints — collaborative action constraints and missing preconditions constraints.

Collaborative Action Constraints

For each collaborative action a_c executed at time t in policy tree τ_i, we add a constraint for all other agents that participate in a_c to also execute the action at time t. However, even though a_c is executed always at time t, in some branches it might be that a_c is not executed at all. Consider the tree in Figure 2. In two branches the agents do not jointly push the heavy box, because it is already at the target position. Hence, the constraint to jointly push the heavy box applies only in branches where the box is not initially at its target position.

The collaborative action constraints are hence *conditional* constraints, conditioned on the value of some observed variables. The collaborative action a_c must be executed only in branches where the value of the observed variables conforms to the conditioned values. To identify these variables we look at the set of branches B_{a_c} where a_c was used, and the set of branches $B_{\neg a_c}$ where a_c was not used.

We identify the set of literals P_{a_c} that occur in all branches in B_{a_c} prior to the execution of a_c in that branch, and the set of literals $P_{\neg a_c}$ that occur in all branches in $B_{\neg a_c}$ throughout the branch execution. Then, $P_{a_c} \setminus P_{\neg a_c}$ defines the difference between these branches. All agents that collaborate on a_c must be able to observe the value of these propositions. Otherwise, the collaborative action cannot be soundly executed.

To employ the constraint in the single agent compiled problem, we implement time into the contingent problems. Although it is inconvenient to implement time into the propositional, PDDL based, description that we use, in our case we implement time only for a limited horizon, equivalent to the depth d of the already computed tree. We set the successor of time t_d to be t_∞, and the successor of time t_∞ to be also t_∞, allowing other agents to plan beyond time d if need be. We add a time parameter to all actions.

We now set the preconditions of all actions except a_c at time t to contain the conditioned variables values. That is, all actions except a_c can be executed at time t only in branches that conform to the conditioned variable values, where a_c was not executed in the original tree τ_i. In the example above, we add to the precondition of all actions at time t_4 the literal $\neg BoxAt_{2,2}$. In order to execute any action other than $JointPushUp_{2,2}$ at time t_4 the agent must first observe the value of $BoxAt_{2,2}$. Although it is not always the case, in this example, the collaborative action $JointPushUp_{2,2}$ has a precondition $BoxAt_{2,2}$, forcing the agent to observe the value of $BoxAt_{2,2}$ before time t_4 in all branches.

3.4 Forward Progression and Backtracking

Given the constraints extracted from the adjusted policy tree for agent i, we now create a new single agent planning problem for the next agent $i+1$. The planning problem augments the definition in Section 3.1 with the collaborative action and missing precondition constraints above. Constraints irrelevant to agent $i+1$, such as missing preconditions that $i+1$ cannot achieve, or collaborative actions that do not apply to $i+1$ are not added onto the single agent problem.

Agent $i+1$ now runs the contingent solver over the constructed single agent planning problem. If a solution is found, then we again level the policy tree, extract additional constraints, and create a new planning problem for the next agent based on all constraints gathered thus far.

Although the original, multi-agent problem may have a solution, the single agent problem of $i+1$ may not be solvable. For example, it may be that agent i added a constraint for agent $i+1$ to be at

Time	t_1	t_2	t_3	t_4
Agent	a_1	a_2	a_1	a_2
$g_1 = \neg BoxAt_1$	1	-	**1**	**-**
$g_2 = \neg BoxAt_2$	4	4	**4**	**4**
$g_3 = \neg BoxAt_3$	6	1	**-**	**1**

Table 1: Iterated improvements of goal completion time for the simple box-pushing problem. Agent a_1 plans alone, achieving goals g_1, g_2, g_3 at t_1, t_4, t_6, respectively, and assuming that a_2 will help pushing Box_2 at t_4. Agent a_2 manages helping pushing Box_2 and achieves g_3 faster at t_1, therefore improving the overall plan and forcing a_1 to replan. Agent a_1 plans without g_3, and a_2 replans ignoring g_1.

position p at time t, but agent $i+1$ is unable to reach p at the appropriate time. To understand why the solver fails, we now plan again for each constraint for agent $i+1$, attempting to achieve the specified constraints by reversed order of appearance. That is, we start removing constraints that occur later in the plan, until a solution is found. This allows us to identify the first constraint c that agent $i+1$ cannot achieve at the required time t_c.

We now replan for agent $i+1$ given all constraints prior to c, where the goal is to achieve c at any time after t_c. If c cannot be achieved at any possible time after t_c, then we fail. Otherwise, the planner computes a plan that achieves all constraints prior to c, and achieves c at time t' at the latest.

We now identify the agent $j < i+1$ that required the constraint c, which agent $i+1$ cannot fulfill in the given time t_c, and report back that the constraint c can be achieved only at time $t' > t_c$. We now backtrack and replan for agent j, with a constraint that c can only be achieved at t' at the earliest. All plans between j and $i+1$ are removed. That is, if j manages to solve the problem, we move to agent $j+1$ and continue.

3.5 Sub-Goal Assignment

While the above process can be used to solve the multi-agent problem, it may produce inefficient plans. Consider, for example, the policy tree generated in Figure 2. The agent produced a solution for pushing all 3 boxes. Obviously, however, it is more efficient to leave the rightmost box at position 3 for agent 2 to handle, as done in the local policy trees in Figure 1. We now describe a mechanism that allows agent 1 to entrust the task of pushing the rightmost box to agent 2.

In addition to the constraints described in Section 3.3, we extract from the policy tree of agent i for each goal literal $g \in G$, the minimal time t_g when g is achieved. We allow other agents to acknowledge that g can be achieved by agent i at time t_g, by adding a conditional effect $(time_{t_g}, g)$ to all actions. That is, every action executed at time t_g achieves g.

An agent j may, however, achieve g at a time $t' < t_g$. In our running example, as shown in Table 1, agent 1 achieves the goal $\neg BoxAt_3$ at time 6. Agent 2 can help agent 1 in pushing the heavy box, and then wait for time 6, but a shorter plan for agent 2 achieves $\neg BoxAt_3$ at time 1, and only then assist agent 1 with the heavy box, at which point all goals have been achieved.

When progressing forward in the agent sequence we maintain for each such goal $g \in G$ only the earliest time of achievement t_g, and add the resulting conditional effects when planning for all agents $j > i$. Once we successfully finish planning for the last agent, we start again from the first agent, this time allowing agents to use all the goal achievement conditional effect. Each agent uses only goal achievement effects created by other agents, to avoid a case where agent i assigns goal g to agent j, while j assigns g to i, and thus no agent actually achieves g.

This process may be repeated several times, because an agent that earlier achieved g at time t_g, may now be able to achieve g at time $t' < t_g$, for example because it now ignores other goals achieved more

Table 2: Comparing IMAP to Dec-POMDP solvers over small, 1D, box pushing problems. $|A_i| = 4$, $|\Omega_i| = 2$. W is the grid width, L and H denote the number of light and heavy boxes. $E[C]$ is computed given a uniform initial belief and unit costs. *P* is the number of single agent planning episodes.

Domain					Compilation		GMAA-ICE		DICEPS		JESP		IMAP		
W	L	H	$\|S\|$	$\|b_0\|$	Time	$E[C]$	Time	$E[C]$	Time	$E[C]$	Time	$E[C]$	Time	$E[C]$	P
2	2	0	16	4	12.8	1.5	**0.22**	**1.5**	16.18	**1.5**	0.78	**1.5**	7.23	4.6	2
3	2	0	36	4	25.5	1.5	**0.58**	**1.5**	18.19	**1.5**	1.67	**1.5**	6.28	4.87	2
3	2	1	70	8	50.3	4.5	×	×	40.88	**3.85**	×	×	**37.07**	4.06	2
5	2	1	200	8	164.6	6.5	×	×	83.4	**4.38**	×	×	**41.84**	12.24	4
5	4	1	800	32	×	×	×	×	×	×	×	×	**123.17**	**14.17**	4

rapidly by other agents. However, each time we start over it is because at least one goal was achieved at an earlier time than in the previous iteration. Hence, this process must terminate eventually, and we repeat this goal allocation replanning process until we converge.

3.6 Ensuring Soundness

There can be several reasons why the above procedure may not produce a sound, executable, plan. For example, it may be that one agent consumes a precondition that another agent relies on. One can simulate the joint policy for every possible initial state, effectively traversing the implicit joint policy tree, but this process is exponential.

We take instead an approximate approach. Each agent collects from all other agents the effects of their actions at each time step. Then, the agent checks whether the preconditions of its local policy tree are not invalidated by the actions of other agents.

Our soundness test is approximate because when an agent produces an effect p in two different branches, at different time steps t, t', where $t < t'$, we assume that p is achieved at time t in all branches. Thus, our soundness test is stronger than needed, and it may be that a valid solution would be discarded, but not vice versa.

4 Experimental Results

We now provide experimental results focusing on scalability to larger QDec-POMDP problems. We experiment with two domains — a variant of the well-known box pushing problem [14], and an adaptation of the rovers domain [17]. All transitions and observations in both domains are deterministic. We use CPOR [10] as the underlying planner, and Metric-FF as the classical planner of CPOR. The experiments were run on a Windows 10 64-bit machine, $i5, 2.2GHz$ CPU, and 8GB RAM. IMAP is implemented in $C\#$, while Dec-POMDP solvers were run on an Ubuntu virtual box on the same machine [1].

4.1 Domains

In the box pushing domain a set of boxes are spread in a grid, and the agents must push each box to a designated location at the edge of the grid (the end of the column it appears in). Each box may be either in a pre-specified location, or at its goal location to begin with. The agent must be in the same location as the box in order to observe where it is. Agents may move in the 4 primary directions, and can push

[1] The implementation of IMAP can be found at https://github.com/Sharpen6/IMAP, and the benchmark domains at https://github.com/Sharpen6/PlanningProblems

boxes in these 4 primary directions, if they occupy the same location as the box. Some boxes are heavy and must be pushed by two agents jointly. There are 9 actions, and 2 possible observations.

We also experiment with an adaptation of the multi-agent rovers domain, where multiple rovers must collect together measurements of soil, and rock. The agents navigate a map of waypoints, and successful measurements can only be taken at some waypoints, unknown initially to the agents. When an agent is at a waypoint it can attempt a measurement, which may be successful or not based on whether the waypoint is appropriate for that measurement. Images of rocks, and samples of soils can be collected by a single rover, while rock samples require two rovers working jointly. After collecting the measurements, the rovers must broadcast them back to the ground station.

4.2 Comparison to Dec-POMDP Solvers

We begin with comparing IMAP to exact (GMAA-ICE [16]) and approximate (JESP [11], DICEPS [12]) Dec-POMDP solvers on small box pushing domains. While IMAP finds satisfying plans, without any optimality criterion, the Dec-POMDP solvers optimize for minimizing expected cost for achieving the goal, denoted $E[C]$. All solvers were executed on various horizons until convergence, and we report the result over the horizon with best $E[C]$ (computed assuming a uniform initial belief and unit costs). In all domains we used a 1D grid, and hence only 4 actions per agent (observe-box, push, move-right, move-left). We also compare to the compilation-based approach [6]. All solvers managed to solve only very small problems, with a short horizon.

We acknowledge that this comparison is not entirely fair, because Dec-POMDP solvers try to optimize solution quality, whereas we only seek a satisfying solution. Thus, Dec-POMDP solvers may need to explore many more branches of the search graph, at a much greater computational cost. Furthermore, many Dec-POMDP solvers are naturally anytime, and can possibly produce a good policy even when stopped before termination. It may well be that solvers may reach a satisfying policy, which is the goal in a QDec-POMDP, well before they terminate their execution.

4.3 IMAP on Larger Problems

Table 3 shows results over large domains. We experimented with varying grid sizes, number of agents, and different compositions of light and heavy boxes . We report runtime (T, secs.), the expected cost under a uniform distribution ($E[C]$), and the plan makespan (M, maximal time to completion), as a measure of plan quality, and the total number of single agent planning episodes (P). The difficulties in this domain are mainly due to the number of collaborations, and the alternative goal assignments.

In the Rovers domain, the problem difficulty is defined, as before, by the number of agents and the size (number of waypoints), but also by the uncertainty — the number of potential waypoints where a measurement can be taken. While the number of states is smaller, the number of actions, and the initial belief uncertainty, are larger in this domain compared to the box pushing problems. The difficulty mostly stems from the size of the initial belief, requiring more complex single agent plan trees.

As can be seen, IMAP scales well to problem sizes well beyond the ability of current planners. The complexity of the problem grows with both the state space size, as well as the number of agents, the number of required collaborative actions, and the initial uncertainty. The number of planning episodes encapsulates both constraints that could not be met, or plan improvements by later agents. Where the number of planning episodes is identical to the number of agents, there was no need to backtrack. The largest number of planning episodes is 12, where 5 agents had to push 3 boxes, and several agents suggested improvements to the initial plan, requiring us to go back and forth. Still, this did not cause the planner to take an exceptionally long time. The main bottleneck of IMAP is the time required to compute a single agent plan in the larger domains, not the number of replanning episodes.

Table 3: IMAP on large domains. *C* is the number of required collaborations, *Time* is the runtime (secs), *M* is the final makespan, $E[C]$ is the expected cost under a uniform distribution, and *Planning* is the number of agent planning episodes.

$\|S\|$	$\|I\|$	$\|A_i\|$	$\|\Omega\|$	$\|b_0\|$	C	Time	M	$E[C]$	Planning
Box pushing									
196	2	8	2	4	1	13.2	16	8	5
400	2	8	2	4	1	30	18	10.2	3
25000	5	8	3	8	1	56.2	13	6.82	12
1000	3	8	3	8	1	59	12	6.68	11
4000	3	8	5	32	2	247.1	19	11.6	9
Rovers									
308	2	15	2	4	1	13.1	18	7.81	2
462	2	16	3	8	1	14.4	16	8.7	2
300	2	16	7	128	1	33.7	35	19.3	2
20250	3	15	5	32	1	37.1	12	5.5	3
1500	3	15	7	128	1	156.3	26	11.2	6
600	2	16	9	512	1	205.01	39	17.6	4

Table 4: IMAP execution example. P denotes the planning episode, BT denotes the reason for backtracking (F - constraint failure, I - goal improvement).

P	1	2	3	4	5	6	6	7	8	9	10	11	12
Agent	1	2	1	2	1	2	3	1	2	3	1	2	3
box_0	4	×	4	4	4	4	×	7	7	7	7	7	7
box_1	14	14	14	5	5	5	5	5	17	4	4	4	4
BT		F		I			F			I			

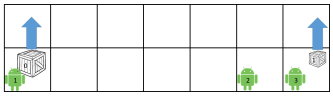
Figure 3 shows an example a box pushing domain with 3 agents and 2 boxes, box_0 which is heavy, and box_1 which is light. Table 4 shows the execution of IMAP on this domain. Agent 1 wants to push box_0 with agent 2 at time 4, and then push box_1 at time 14. Agent 2 reports that it can only push box_0 at time 7. Agent 1 now selects agent 3 for collaboration. Agent 2, relieved of pushing box_0 can push box_1 at time 5. Agent 1 confirms this plan improvement. When reaching agent 3 it reports that it can help with box_0 only at time 8. Agent 1 replans, and chooses collaboration with agent 2, who cannot push box_1 at time 5. Agent 3 now reports that it can push box_1 at time 4. All agents confirm the new plan.

5 Discussion and Related Work

Our approach is well rooted in the multi-agent literature. Iterating over the agents, focusing on one agent at a time is used for reducing the complexity of considering a joint policy. For example, JESP [11] modifies the policy of one agent, while keeping the policies of all other agents fixed. The exponential complexity of considering all possible joint observations is not reduced, though, making scaling up difficult, as can be seen in our experiments as well.

Similar approaches have been used in other multi agent problems such as DCOP [7], privacy preserving planning [3], and many more. Focusing on interaction points between the agents is also a well known idea. In Dec-POMDPs, [15] reduce the problem complexity by considering states in which agents must interact, and states where they can act independently. Similar intuitions were pursued by [18] and [13], for decoupling the state space considering how agents influence one another. To resolve the dependencies, the AI community has suggested that agents should enforce commitments [8]: constraints on policies that must be adhered. In multi agent planning, these commitments take a very similar role and structure to what we do [4].

Figure 3: Box pushing example for Table 4



As such, the main contribution in this paper is in an efficient adaptation of these well known ideas into the new QDec-POMDP framework, scaling well beyond the ability of current exact and approximate Dec-POMDP solvers.

6 Conclusion

We presented IMAP — an iterative algorithm for multi-agent planning for QDec-POMDPs, which iteratively plans for a single agent. IMAP assumes that other agents will be available to help the agent with collaborative actions, and produces constraints to the other agents for the required collaborations. Later agents that cannot meet these constraints results in backtracking. IMAP also allows for later agents to improve the previous plans, by taking responsibility for some tasks, causing again backtracking.

We experiment with two types of domains, the well-known box pushing domains and a new Dec-POMDP domain adapted from the multi-agent planning community. On both domains, we have shown a scaling up ability well beyond the limitation of current Dec-POMDP planners.

In the future, we will experiment with more domains, focusing on other types of collaborations. E.g., we would explore cases where each agent completes a part of a task, but there are no collaborative actions. We will also explore cases where agents execute actions that interfere with other agents, such as actions that consume a precondition required by another agent. We believe that simple extensions of our approach will be able to handle such cases.

An important extension of our algorithm is for non-deterministic actions, incurring loops. Our constraints must be significantly modified to cope with loops.

Acknowledgements

This work was partially supported by the CBG Cyber Security Center at Ben Gurion University of the Negev, and by ISF grant 933/11.

References

[1] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27:819–840, 2002.

[2] Blai Bonet and Hector Geffner. Belief tracking for planning with sensing: Width, complexity and approximations. *J. Artif. Intell. Res.*, 50:923–970, 2014.

[3] Daniel Borrajo. Multi-agent planning by plan reuse. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1141–1142. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[4] Ronen I Brafman and Carmel Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS*, pages 28–35, 2008.

[5] Ronen I. Brafman and Guy Shani. Online belief tracking using regression for contingent planning. *Artif. Intell.*, 241:131–152, 2016.

[6] Ronen I. Brafman, Guy Shani, and Shlomo Zilberstein. Qualitative planning under partial observability in multi-agent domains. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA.*, 2013.

[7] Archie C Chapman, Alex Rogers, Nicholas R Jennings, and David S Leslie. A unifying framework for iterative approximate best-response algorithms for distributed constraint optimization problems1. *The Knowledge Engineering Review*, 26(4):411–444, 2011.

[8] Nick R Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The knowledge engineering review*, 8(3):223–250, 1993.

[9] Radimir Komarnitsky and Guy Shani. Computing contingent plans using online replanning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 3159–3165, 2016.

[10] Radimir Komarnitsky and Guy Shani. Computing contingent plans using online replanning. In *AAAI*, pages 3159–3165, 2016.

[11] Ranjit Nair, Milind Tambe, Makoto Yokoo, David Pynadath, and Stacy Marsella. Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In *IJCAI*, volume 3, pages 705–711, 2003.

[12] Frans A Oliehoek, Julian FP Kooij, and Nikos Vlassis. The cross-entropy method for policy search in decentralized pomdps. *Informatica*, 32(4), 2008.

[13] Frans Adriaan Oliehoek, Stefan J Witwicki, and Leslie Pack Kaelbling. Influence-based abstraction for multiagent systems. In *AAAI*, 2012.

[14] Sven Seuken and Shlomo Zilberstein. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 344–351, Vancouver, British Columbia, 2007.

[15] Matthijs TJ Spaan and Francisco S Melo. Interaction-driven markov games for decentralized multiagent planning under uncertainty. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 525–532. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[16] Matthijs TJ Spaan, Frans A Oliehoek, and Christopher Amato. Scaling up optimal heuristic search in dec-pomdps via incremental expansion. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 2027, 2011.

[17] Michal Stolba, Antonín Komenda, and Daniel L. Kovacs. Competition of distributed and multiagent planners (codmap). In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 4343–4345, 2016.

[18] Stefan J Witwicki and Edmund H Durfee. Influence-based policy abstraction for weakly-coupled dec-pomdps. In *ICAPS*, pages 185–192, 2010.