Signaling in 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. Although such problems are often modeled as Dec-POMDPs, recent work has demonstrated that planners for *Qualitative* Dec-POMDPs can scale up to much larger problem sizes. The best current QDec-POMDP solver is based on a *factored planning* (QDec-FP) algorithm in which the multi-agent problem is reduced to multiple single-agent problems. In this paper we describe how we augment this approach with the ability of agents to signal information by changing the state of the world (e.g., turning on a *light* to signal whether a door is open). To support signaling, the QDec-FP planner must model and reason about the knowledge of each agent separately. Our modified version of the QDec-FP algorithm augments the translation used with propositions representing individual agents' knowledge, and adds preprocessing steps that augment the domain with signaling macros, and post-processing steps that utilize these macros. The resulting algorithm supports signaling, and also greatly improves the planner's ability to handle problems with multiple objects.

1 Introduction

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

In this setting, it is common to compute a policy 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 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-POMDP 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). QDec-POMDPs can also be viewed as the multi-agent extension of the well known Contingent Planning model (Hoffmann and Brafman, 2005). At least for deterministic problems, where partial observability plays the key role, QDec-POMDP algorithms scale much better than Dec-POMDP algorithms. This was demonstrated clearly by the IMAP solver (Bazinin and Shani, 2018), which was later improved by using factored planning techniques (Shekhar, Brafman, and Shani, 2019).

The factored planning algorithm (QDec-FP) (Shekhar, Brafman, and Shani, 2019) starts by solving an abstract and simplified centralized problem which we call the *team problem.* The team problem is similar to the original MAP problem, but assumes centralized execution, where all agents are controlled at execution time by a single meta-agent that has access to all their observations. The solution to the team problem serves as the skeleton for the true MA solution. To extend this skeleton to a complete solution, each agent needs to (independently) ensure that it can carry out its part in the skeleton solution. Thus, each agent may extend the skeleton with additional actions. If all agents can complete their part of the skeleton, a final step of aligning the plans of the agents is carried out. If some agent cannot complete its part, the algorithm backtracks and must seek a new team solution.

QDec-FP, treats the team planning problem as a single-agent problem and uses a single-agent contingent solver to solve it. It does not keep track of the knowledge acquired at run-time by each agent separately. Therefore, the team plan may require that agent φ_j act differently based on the value of some proposition p observed by agent φ_i, even if φ_j cannot observe p. While the second stage of QDec-FP will fix this problem or rule out this solution, it would be much more efficient if QDec-FP incorporated reasoning about the knowledge of individual agents during team planning. This has two important advantages: First, it will lead to the generation of more informed team plans that are easier to extend. Second, it allows us to model, within the team plan, the process of explicit and implicit communication, which we refer to here as *signaling.*

Signaling refers to a situation where agent φ_i notifies agent φ_j of the value of some attribute that φ_j cannot observe, by manipulating another attribute that φ_j can observe. For example, suppose agent φ_j cannot sense whether a door is open, but it can sense whether the light is on, while agent

φ_i can sense both. If φ_i can also turn the light on and off, it can signal the door state to φ_j by turning on the light if and only if the door is open.

More generally, signaling handles cases where φ_i can sense p but φ_j cannot. Agent φ_i might be able to communicate this information to φ_j directly, if a communication action exists, or indirectly, by making sure that the value of some variable q that agent φ_j can sense, is correlated with the value of p. Technically, signaling consists of the following steps: (1) Agent φ_i senses p. (2) Agent φ_i sets the value of q to the value of p. (3) Agent φ_j senses q. To be sound, this behavior must be consistent between the two execution branches that follow the sensing of p: if p is *true*, we must ensure q is *true*. If p is *false*, we must ensure q is *false*.

Some problems cannot be solved without signaling, while others can be solved more efficiently with signaling. To support signaling in QDec-FP, we must first address two key technical issues. First, we must explicitly account for the knowledge of different agents. Currently, when the planner generates a team solution, it assumes centralized execution and joint knowledge of all agents. Therefore, signaling actions have no impact on the state, and will not be inserted into the team plan. We address this by explicitly modeling the knowledge of each agent, and allowing an action in the team plan only if the executing agent can verify that its preconditions are true. Aside for supporting the generation of plans with signals, this more accurate model of the MA system prunes many plans that were previously generated by the team planner, yet are not extendable to MA plans. This reduces the need for backtracking between team plans, but places a heavier burden on the computation of the team plan.

In addition, we must consider the implementation of signaling. Signaling must ensure correspondence between the signaled proposition and the signaling proposition, which is not a local constraint. That is, if we want to signal p's value using q, we must ensure a correlated value for q both in the branch of the plan in which p is *true* and in the branch in which p is *false*. Because this is a constraint over the plan-tree, rather than its branches, supporting it is more complicated. We solve this by adding a signaling macro that is post-processed after the team-plan generation. Intuitively, one can understand this macro as expressing a commitment by one agent to make sure that from this point-on, p and q will be correlated. This allows other agents to deduce the value of one proposition from that of the other.

In this paper we describe a new algorithm, QDec-FPS, which provides both support for reasoning about the individual knowledge of agents and exploits it to support signaling. We also provide an initial empirical evaluation of QDec-FPS and show that it allows us to solve problems that were not solvable by previous methods, because they require that some agents must act differently given different values of a variable they cannot observe, as well as to scale up the QDec-FP approach to much larger problems even when signaling is not required.

2 Background

We describe the QDec-POMDP model, the structure of its policies, and the QDec-FP solver.

2.1 QDec-POMDPs

We define a flat state space QDec-POMDP, followed by a factored definition motivated by contingent planning model definitions (Bonet and Geffner, 2014).

Definition 2.1. A qualitative decentralized partially observable Markov decision process (QDec-POMDP) is a tuple $\mathcal{Q} = \langle I, S, b_0, \{A_i | i \in I\}, \delta, \{\Omega_i\}, \omega, G\rangle$ 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.
- $\delta : S \times \vec{A} \to 2^S$ 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.
- $\omega : \vec{A} \times S \to 2^{\vec{\Omega}}$ 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, 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 natural for an off-line centralized algorithm.

We focus on a factored representation of a QDec-POMDP, specified as follows: $\langle I, P, \{A_i | i \in I\}, Pre, Eff, Obs, b_0, G\rangle$ where I is a set of agents, P is a set of primitive propositions, $\vec{A}$ is a vector of individual action sets, *Pre*, *Obs*, and *Eff* are the precondition, observation, and effects functions, b_0 is the initial state formula, 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. 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, Shani, and Zilberstein (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 and not a simple union of effects of local actions, due to possible interactions between the different local actions. However, in line with Bazinin and Shani (2018); Shekhar and Brafman (2018), we employ a more structured definition of joint actions.

We assume that single-agent actions (of a joint action) 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, as modeled in (Shekhar and Brafman, 2018). If a collaborative action such as *joint-lift* exists, and a 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. When concurrent actions that delete another agent's preconditions or object capacity constraints (Crosby, Jonsson, and Rovatsos, 2014) are not allowed, 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 a deeper discussion of the definition of joint actions, see (Shekhar and Brafman, 2018).

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

In this paper we assume that actions either sense or change the state of the world. Sensing actions do not affect the world state, i.e., if a is a sensing action then $\mathit{Eff} = \emptyset$. Non-sensing actions do not provide any information, that is, $Obs(a) = \emptyset$. This is not a limiting assumption, as every action can be separated into a non-observation and a sensing action by adding suitable propositions forcing the two to appear consecutively in every plan. Furthermore, we assume that any agent has a *noop* action, with $Pre(noop) = \mathit{Eff}(noop) = Obs(noop) = \emptyset$.

While QDec-POMDPs allow for non-deterministic effects and 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.

Another important concept in multi-agent planning is the notion of public and private actions and propositions (Brafman and Domshlak, 2008). A proposition that appears in the action descriptions of multiple agents is called *public*. Also, all goal propositions are public. An action is a *public action* if its precondition or effect contain a public proposition. A proposition that appears only in the actions of agent φ_i is called *private* to φ_i. An action whose preconditions and effects contain only private propositions is a *private action*.

2.2 Policy Trees

We can represent the local plan of an agent φ_i using a *policy tree* τ_i. Each node of the tree is labeled by an action, and edges that follow a sensing action 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. For a policy tree τ_i for agent φ_i, and a possible observation o_i for φ_i, $\tau_{i_{o_i}}$ denotes the child subtree 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 $\vec{\tau}_{\vec{o}} = \langle \tau_{1_{o_1}}, \ldots, \tau_{m_{o_m}}\rangle$.

As 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 can maintain the set of states possible at each node during the execution of the joint policy, typically called the *belief state*. One must distinguish between the belief state of the entire system and the belief of a single agent. Online, each agent may have less information, as it does not know of the observations of other agents. Hence, it cannot distinguish between all branches of the joint-policy. We denote by $\vec{b}$ the set of agent-specific beliefs. When computing the team plan, we are assuming a centralized execution engine and a centralized belief.

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 $b \models pre(\vec{a}_{\vec{\tau}})$. In that case, we say that $\vec{a}_{\vec{\tau}}$ is *executable* in b. After the agents execute $\vec{a}_{\vec{\tau}}$ and observe $\vec{o}$, their new belief state is $tr(b, \vec{o}, \vec{a}_{\vec{\tau}}) = \{\vec{a}_{\vec{\tau}}(s) | s \in b, \vec{a}_{\vec{\tau}}(s) \models \vec{o}\}$.

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 φ_j is another agent participating in that collaborative action, then a_{τ_j} contains φ_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 the belief state in all leaf nodes in the tree satisfies G. 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 marked 1-3, and two agents φ_1 and φ_2, 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*

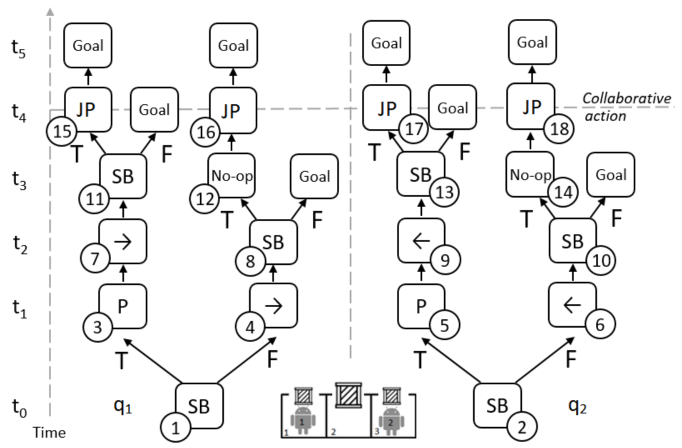

Figure 1: Illustration of Example 1 showing the box pushing domain with 2 agents and a possible set of local plan trees that produce a solution. Possible agent actions are sensing a box at the current agent location (denoted SB), moving (denoted by arrows), pushing a light box up alone (denoted P), jointly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

single agent may push them alone. The middle box is heavy, and requires that the two agents push it together.

Here $P = \{AgentAt_{i,pos}, BoxAt_{j,pos}, Heavy_j\}$ *where* $pos \in \{1,2,3\}$ *is a possible position in the grid,* $i \in \{\varphi_1, \varphi_2\}$ *is an agent, and* $j \in \{1,2,3\}$ *is a box index. In the initial state each box may be in or out of the grid —* $b_0 = AgentAt_{\varphi_1,1} \wedge AgentAt_{\varphi_2,3} \bigwedge_{j=1,2,3}(BoxAt_{j,j} \vee \neg BoxAt_{j,j})$*. 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 an agent* φ *to push up a light box* j*, agent* φ *must be in the same place as the box. That is,* $Pre(PushUp_{\varphi,j}) = \{AgentAt'_{\varphi,j}, BoxAt_j, \neg Heavy_j\}$*. For the collaborative joint push action the precondition is* $Pre(JointPush_j) = \{AgentAt_{\varphi_1,j}, AgentAt_{\varphi_2,j}, BoxAt_j, Heavy_j\}$*.*

The moving actions transition the agent from one position to the other, and are independent of the effects of other agent actions, e.g., $Right_\varphi = \{(AgentAt_{\varphi,1}, \neg AgentAt_{\varphi,1} \wedge AgentAt_{\varphi,2}), (AgentAt_{\varphi,2}, \neg AgentAt_{\varphi,2} \wedge AgentAt_{\varphi,3})\}$*. The only joint effect is for the JointPush action –* $Eff(PushUp_{\varphi,2}, a_2)$ *where* a_2 *is some other action, are identical to the independent effects of action* a_2*, while* $Eff(PushUp_{\varphi_1,2}, PushUp_{\varphi_2,2}) = \{(\phi, \neg BoxAt_{2,2})\}$*, 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 — $SenseBox_{\varphi,j}$*, with precondition* $Pre(SenseBox_{\varphi,j}) = AgentAt_{\varphi,j}$*, no effects, and* $Obs(SenseBox_{\varphi,j}) = BoxAt_{j,j}$*. 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* to denote a tree that is obtained from the original policy tree by removing some nodes. When a node n labeled by a non-sensing action is removed, the parent of n becomes the parent of the child of n. A node n labeled by a sensing action can only be removed if the two child subtrees of n are identical. In that case, the parent of n becomes the parent of one of the identical subtrees.

3 The QDec-FP Algorithm

QDec-FP (Shekhar, Brafman, and Shani, 2019) operates in three stages. (1) Generating a team solution skeleton. (2) Extending the projection of the team solution for each single agent. (3) Aligning the single agent plan trees.

In the first step, the MA problem is transformed into a single-agent problem by treating all actions as if executed by the same meta-agent. This meta-agent can apply any agent's action and and it is aware at each stage of the results of all sensing actions. We denote the resulting team plan as τ_{team}.

In the second step, from τ_{team}, QDec-FP generates for each agent φ_i a projection τ_{φ_i}. To obtain the projection we first remove from the tree all non-sensing actions except those executed by agent φ_i. Second, we remove from the tree all private actions of all agents, leaving only nodes labeled by public actions. Furthermore, for each action a in τ_{φ_i}, we remove all its public preconditions, as they are guaranteed to be supplied by some public action in τ_{team}. Finally we remove redundant sensing actions — observations that do not influence how φ_i acts. These are sensing actions where both subtrees below the observation are identical, from the perspective of agent φ_i.

An example of a team solution, its projection, and the compacted projection is given in Figure 2. In Figure 2A, we can see the team solution, containing both private and public actions of the two agents. The team plan assumes shared knowledge. We can see, e.g., that in the left branch, only agent φ_1 senses for the existence of the heavy box, and then the agents jointly push it. Figure 2B shows the projected tree of φ_1. All private actions of both agents were removed, and all sensing actions of both agents remain. Here, no sensing action was redundant. Figure 2C shows the projected tree of φ_2. Here, also the public push action executed by φ_1 in the team plan is removed. Furthermore, as φ_2 operates identically in both subtrees of the team plan, we remove the sensing action at the root of the team plan.

The projected tree is typically not executable by the agent. It contains sensing actions that are executed by other agents, and some actions require additional preconditions, due to the removal of private actions. Next, QDec-FP solves a single-agent contingent planning problem for each agent φ_i that requires φ_i to make τ_{φ_i} executable by adding sensing actions and private actions. QDec-FP does not add public actions, as they could interfere with the policies of the other agents by, e.g., consuming a precondition that is needed.

If some τ_{φ_i} is not solvable, QDec-FP backtracks and seeks a new team solution. If all τ_{φ_i} are solvable, we know that there is a solution, and all that remains is to align the different policies to ensure that agent actions are executed in the right order – as reflected by the team policy. This is done by inserting *noop* actions to postpone actions as need be.

While the above is a general scheme, it requires an underlying single-agent off-line contingent planner, i.e., a contin-

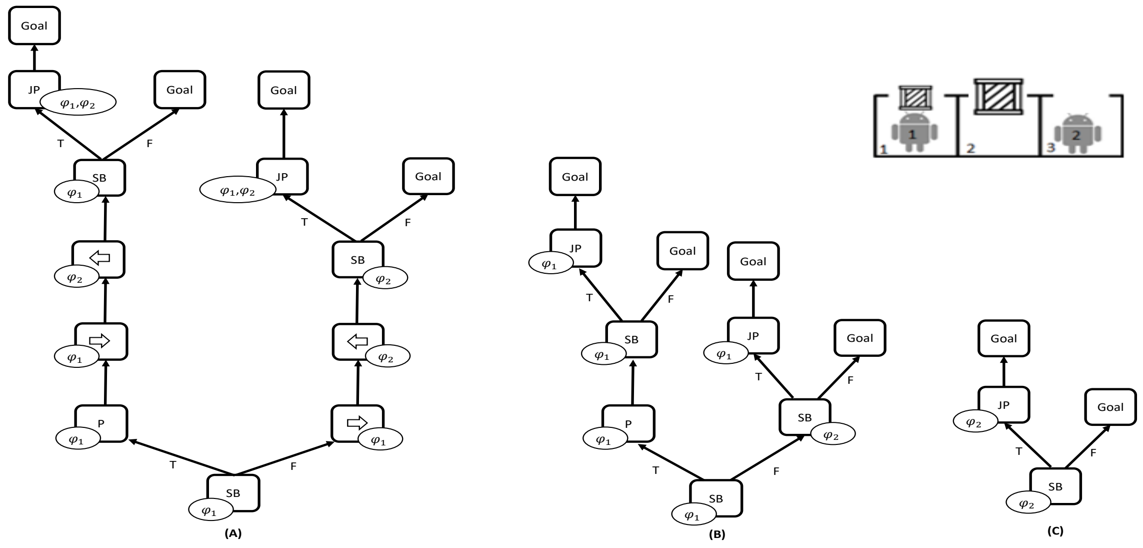

Figure 2: (A) Team plan tree τ_{team} for a problem with two agents, a light box and a heavy box that need to be outside at the edge of the grid in the goal state. (B) The projection of τ_{team} to φ_1 for which all the sensing actions of τ_{team} are non-redundant and remain. (C) A compacted projection for φ_2 in which no sensing action by φ_1 is required.

gent planner that generates complete policies. For this purpose we use the CPOR contingent planner (Komarnitsky and Shani, 2016). CPOR itself calls an online contingent planner, SDR (Shani and Brafman, 2011). SDR generates a single execution branch that corresponds to the true initial state. In the case of CPOR, a different "true" initial state is selected by the algorithm until all states have been covered.

While CPOR is the most scalable offline contingent planner currently, it does have a major weakness — it does not perform backtracking. QDec-FP implemented a certain backtracking ability on top of it by modifying the planning problem with an additional, sound, constraint, so that the previous solution cannot be generated again. However, this limits the number of backtracks that are practically possible.

The reliance on an online solver that focuses on one branch at a time has important implications for the QDec-FP implementation. The SDR solver is not aware of the choices made in alternative branches. This can lead to a subtle issue when reasoning about the knowledge of different agents, as we later discuss in the case of signaling.

4 The QDec-FPS Planner

We now describe our improved version of the QDec-FP planner, which we call the QDec-FPS (that allows for *Signaling*) planner. The QDec-FPS algorithm has identical high-level structure as QDec-FP, but modifies the way the team problem is solved, and adds a pre-processing mechanism that generates macros that enable signaling. While the ideas underlying this extension — modeling the knowledge of different agents and the use of signaling — are general, we also explain the particular technical mechanisms required for integrating these ideas into QDec-FPS.

4.1 Agent Specific Knowledge

QDec-FP uses, through the CPOR planner, the SDR translation from contingent planning to classical planning. This translation maintains, for each proposition p, two propositions: Kp and $K\neg p$, denoting knowing that the value of p is true or false, respectively. SDR then transforms each precondition p of an action to Kp. That is, in addition to requiring that the action will be executable only when p holds, the planner also requires that the agent will *know* that p holds. The agent can obtain Kp as a result of a sensing action over the value of p.

The translation ensures that Kp holds in a belief state only if p holds in all possible worlds. This is done by reasoning about all the specific possible worlds explicitly. The state description in the classical translation contains the current state of the world given every possible initial state, in the form of $Kp|s$, where p is a proposition, and s is a possible state. The translation also contains actions that allow deducing new knowledge facts, called *merge* and *refutation* actions (Palacios and Geffner, 2009). Thus, the agent can also obtain Kp if for every currently possible state s, $Kp|s$ holds.

As QDec-FP solves the team problem as a single-agent planning problem, the planner treats all agents as part of a single centralized agent, and the knowledge of this agent reflects the combined knowledge of all agents. Thus, when one agent observes that p holds, other agents can use this knowledge, without observing the value of p themselves. This is inconsistent with the real plan tree, where each agent must independently ensure that p holds, before executing an action that requires p as precondition.

In QDec-FPS, we augment the translation that is used by SDR to be agent aware. Instead of propositions of the form Kp denoting combined knowledge, the augmented translation uses propositions of the form $K_{\varphi}p$, denoting that only agent φ knows that p holds. Now, the precondition p of an action that is executed by agent φ is replaced in the augmented translation by $K_\varphi p$. The effect of a sensing action executed by φ is that only the sensing agent φ knows the value of p, i.e., either $K_\varphi p$ or $K_\varphi \neg p$. The same holds for the merge and refutation actions, which now provide agent-specific knowledge.

This modification forces the underlying planner to insert sensing actions by different agents to ensure they have the knowledge required to perform their actions, whereas in QDec-FP, because all agents are treated as one, it was enough if some other agent performed this sensing action. If an agent has an action with a precondition p, the team plan will ensure that the agent first senses or learns the value of p. If the agent cannot sense or learn the value of p, such an action will not be part of a generated team plan.

This leads to the generation of skeleton team plans that better account for agent abilities, and are therefore less likely to lead to a failure when each agent extends the skeleton plan. Consequently, it reduces the number of needed backtracks. This team plan, however, may not be executable in a distributed manner. This is because the team plan may require agents to execute different actions under different branches, among which they cannot differentiate. This requires additional sensing actions, which will be added later.

In addition, the new translation allows us to solve problems that QDec-FP could not. Consider, for example, a problem where the values of p and q are always correlated. Agent φ_1 can observe only p, and agent φ_2 can only observe q. If φ_2 needs to execute an action with precondition p, QDec-FP might add the action of sensing p by φ_1 in the team plan. This team plan cannot be later extended by φ_2 to a valid local plan. QDec-FPS would correctly avoid this team plan and consider team plans where φ_2 observes q, and then reasons about p.

The new translation is more demanding and generates classical planning problems that are harder to solve. Because of this, it is able to detect unsolvable problems faster. On the other hand, in the special case of agents with identical capabilities, there are problems that are solvable by QDec-FP but not by QDec-FPS. In these problems, QDec-FP generates a simple team solution that lacks many sensing actions. But adding them in the projected single-agent planning problems is easy because all agents can apply all actions. However, this simple team solution of QDec-FP is not a legal solution to the more complex team problem QDec-FPS generates. This, more complex problem, requires numerous backtracks to solve, making it unsolvable in the given time.

4.2 Signaling

If agent φ_i can sense p but agent φ_j cannot, φ_i might be able to communicate the value of p to φ_j. It could do this directly, if an explicit communication action exists, or indirectly, by setting the value of some variable q that agent φ_j can sense, to be correlated with the value of p. In fact, explicit communication can be viewed as correlating the value of a channel variable with p. Signaling consists of the following steps (1) Agent φ_i senses p. (2) Agent φ_i sets the value of q to the value of p. (3) Agent φ_j senses q. (4) φ_j reasons about the value of p.

Notice that (2) is not a restriction on a single branch of the plan, but a restriction on a sub-tree. φ_i must ensure that $p \leftrightarrow q$, which means that it needs to act differently in the branch where p is true and in the branch where p is false.

To operationalize this idea, we suggest to model the signaling process as a macro. In our context, macros are not simply a sequence of actions, but rather a part of a sub-tree.

To construct such macros, we first must discover the possible signaling options. We preprocess the domain seeking qaudruples $(\varphi_i, p, \varphi_j, q)$ such that (1) φ_i can sense p but φ_j cannot. (2) φ_j can sense q. (3) φ_i can modify the value of q. For simplicity, we consider only propositions q that can be affected by a single action that does not change the value of any other public proposition. This can be extended to more complex sub-plans for modifying the value of q.

For each such quadruple, we add the macro-action $signal(\varphi_i, p, \varphi_j, q)$. This macro is treated by the planner similar to a sensing action, which has two possible outcomes: in one of them $p \wedge q$ holds, and in the other $\neg p \wedge \neg q$ holds. Unlike the pure sensing actions we use, this action changes the state of the world, as well, ensuring this correspondence between the values of p and q.

Given a team plan with a signaling macro, we first expand the macro as indicated above: First, φ_i senses the value of p. For each of the two resulting branches, it must ensure that q's value is appropriately correlated, by applying actions that affect q's value appropriately. Then, we add in each branch a sensing action a_q where φ_j senses the value q. While regular sensing actions have two possible children, a_q has only a single child in the team plan because, at the team level, its outcome is known. When projected to φ_j's local plan, however, a_q appears like a regular sensing action. This macro expansion is described in Figure 3.

In the next step, the projection of the team plan containing the macro expansion is solved by each agent. This requires, in particular, that agent φ_i will change the value of q as needed in each branch, and that both agents perform their sensing actions — φ_i over p and φ_j over q.

Figure 3 shows an example of this process, where two agents, φ_1 and φ_2 must jointly push a heavy box. φ_1 can sense the box, but φ_2 can only sense whether a light is on. φ_1 must hence signal to φ_2 about the box by turning on the light, which is originally off. Figure 3A shows the inserted macro into the team plan. Figure 3B shows the team plan after the macro expansion — φ_1 senses the box, and then turns on the light if needed to be, and then φ_2 senses the light, and they both jointly push the box. Figure 3C shows the projected single agent plan trees for φ_1 and φ_2. Note that the tree for φ_2 contains two children for the *sense-light* action, although the team plan does not. This special case is handled in the projection creation for which a time tick (T) is shown in the right (just for a reference).

In practice, adding this macro on top of an online contingent solver that focuses on a single branch at a time, such as SDR, is not straightforward. To address this, we do the following: We add an action by φ_i that can be viewed as a commitment to ensure that $p \leftrightarrow q$ holds. This action is constrained to be followed immediately by the action of sensing p by φ_i. At this point, in the team plan, if agent φ_j needs to

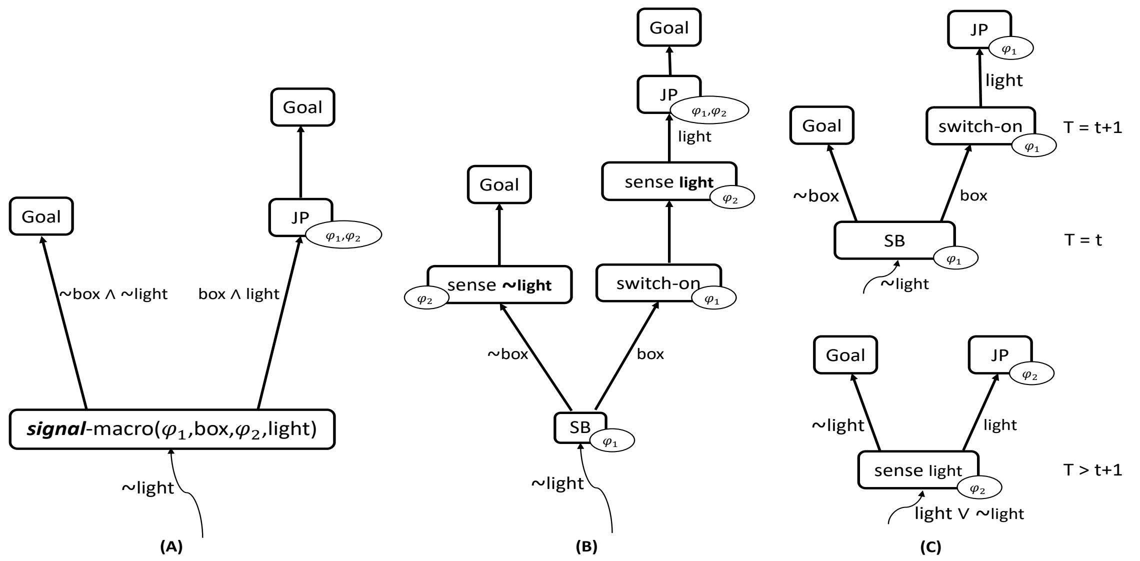

Figure 3: *Signaling* in the QDec-FPS planner. (A) A team plan with the signaling macro action. (B) The team plan following the macro expansion. (C) Projected trees for φ_1 (top) and φ_2 (bottom).

know the value of p, it can use the fact that $p \leftrightarrow q$ to deduce it from the value q. To ensure it learns the value of p, we force the action of sensing q in both branches. As above, the team plan is post-processed to ensure that φ_i does indeed ensure the validity of $p \leftrightarrow q$ following the sensing action.

QDec-FPS Properties. Due to lack of space, we do not discuss soundness and completeness in depth. We briefly note that soundness follows the same lines as that of QDec-FP. With signaling, this also requires that the knowledge reflected by the signaling macro is modeled correctly and that no other proposition is impacted by it.

QDec-FPS is incomplete due to CPOR's lack of backtracking mechanism. But assuming a contingent solver that is able to enumerate solutions, instead, a similar completeness argument to QDec-FP holds. We note that signaling does not make the planner theoretically more powerful, in that case, but it does make it more powerful in practice.

5 Empirical Evaluation

We now demonstrate the scalability and applicability of our new approach by comparing QDec-FPS with QDec-FP on the Box-Pushing and Table-Mover domains. Both the QDec-FP and QDec-FPS solvers are implemented in C#, and were run on a Windows 10, 64 bit machine with i7 processor, 2.8GHz CPU, and 16Gb RAM. It times out after 10 minutes. IMAP was excluded as QDec-FP was already shown to scale better than IMAP (Bazinin and Shani, 2018).

5.1 Domain Description

We consider two domains:

Box-Pushing (BP): There are boxes situated in a grid-like structure. Each box is supposed to moved to its destination location, in this case at the edge of the grid, i.e., at the end of the column the box appears in. Each box is either at some location in the grid or at the goal location. An agent needs to be in the same grid cell where a box exists to observe it and to push it. Boxes are either heavy or light. Two agents are required to push a heavy box successfully, while a single agent can push a light box. Agents can also move between two adjacent locations in the four primary directions. In this domain, we have uncertainty about the initial locations of the boxes. The agents are non-homogenous — different agents can observe and push different boxes.

Table-Mover (TM): The domain consists of a number of tables and rooms, and agents that can move between connected rooms. The initial location of each table is uncertain, and agents must move each table to its dedicated goal locations. Like BP, agents in TM are *non-homogeneous*. Different agents can sense the location of different tables. Each table has some fragile items on top of it, and these objects must remain intact. To achieve this, agent must perform collaborative actions to manipulate the table, for example, collaborative actions like *2move-table*, *2lift-table*, *2drop-table*. The actions *2lift-table* and *2drop-table*, indicate that two agents, respectively, lift and drop a table simultaneously, keeping the objects on the table intact in the process.

5.2 Experimental Results

We compare the QDec-FPS and QDec-FP solvers on the basis of the policy quality (*max-width*, *max-height*), runtime (*time*), and the number of *backtracks* they require to solve a contingent planning problem. *max-width* and *max-height* refer to maximum number of branches and the maximum height of all individual solution trees obtained for the agents. The number of branches is also indicative of the number of sensing actions performed, as branching occurs following an observation. The planner backtracks when at least one of the single-agent problems, obtained by decomposing τ_{team}, is unsolved by CPOR.

Domain	Ins (#agt)	Objects	Max-width		Max-height		Time (sec)		BT	
			fp	*fps*	*fp*	*fps*	*fp*	*fps*	*fp*	*fps*
	B1 (3)	16	8	5	23	18	3.59	**2.91**	0	0
	B2 (4)	16	12	10	19	19	**5.3**	6.1	0	0
	B3 (9)	36	64	*	24	*	**25.3**	*	0	*
	B4 (3)	12	6	8	16	15	13.65	**2.9**	4	0
	B5 (3)	11	4	4	14	11	16.39	**1.17**	9	0
	B6 (3)	12	-	6	-	12	-	**13.58**	47^+	4
	B7 (3)	12	8	8	18	19	158.89	**3.87**	41	0
BP	B8 (3)	12	8	8	17	21	111.6	**4.05**	26	0
	B9 (3)	13	16	14	21	19	121.42	**5.6**	19	0
	B10 (3)	16	16	15	26	29	155.83	**9.69**	33	0
	B11 (5)	20	*	24	*	32	*	**75.21**	*	1
	B12 (5)	20	*	24	*	37	*	**365.9**	*	6
	B13 (2)	12	*na*	2	*na*	2	*na*	**1.2**	*na*	1
	B14 (3)	13	*na*	4	*na*	7	*na*	**5.5**	*na*	1
	B15 (3)	13	*na*	4	*na*	6	*na*	**8.9**	*na*	2
	B16 (4)	15	*na*	8	*na*	14	*na*	**17.6**	*na*	2
	T1 (3)	10	8	7	20	16	**2.68**	2.78	0	0
	T2 (4)	12	16	13	21	17	**7.84**	8.26	0	0
	T3 (4)	15	12	16	34	26	**8.74**	12.39	0	0
	T4 (5)	14	19	22	22	24	**20.5**	43.58	0	0
	T5 (5)	16	12	14	32	25	**9.7**	11.2	0	0
	T6 (3)	8	2	2	8	8	11.01	**0.61**	7	0
	T7 (3)	10	8	7	20	16	**34.5**	41.8	8	8
TM	T8 (3)	10	8	8	24	22	37.87	**22.45**	9	4
	T9 (3)	10	-	-	-	-	140.32	**29.73**	31	6
	T10 (4)	12	16	16	23	22	**6.68**	152.18	0	14
	T11 (5)	16	16	16	30	35	274.5	**56.15**	27	3
	T12 (2)	10	*na*	2	*na*	9	*na*	**1.7**	*na*	0
	T13 (3)	12	2	2	16	11	17.59	**2.32**	9	0
	T14 (2)	12	*na*	4	*na*	17	*na*	**6.81**	*na*	0
	T15 (3)	12	*na*	4	*na*	19	*na*	**11.68**	*na*	1

Table 1: Performance comparison of QDec-FP and QDec-FPS planners. *Ins (#agt)* is instance number with the number of acting agents in the brackets. *Object* denotes the number of objects considered in each problem. Column BT is for the number of times a planner backtracks to solve a problem. * is *time out*, '**-**' represents the planner finds it unsolvable, and *na* is not applicable to a solver (when signaling is required). *fp* is QDec-FP and *fps* is QDec-FPS. The best approach is shown in **bold**, selected solely on the basis of *time*.

Table 1 compares the performance of our QDec-FPS planner with QDec-FP. The table is divided in two major parts, one for each domain. Within each domain, dashed lines separate three problem classes: homogeneous agents, non-homogeneous agents, and non-homogeneous agents that require signaling to solve a problem.

In general, QDec-FPS scales much better than QDec-FP in problems with three up to five agents. An increase in the number of objects that usually makes problems more complex had minor impact on the running time of QDec-FPS as opposed to QDec-FP. For many problems, QDec-FPS needs to backtrack much fewer times than QDec-FP, and as a result, QDec-FPS solves the problems much faster.

Increasing the number of agents has an adverse effect on the performance of the QDec-FPS planner, as the new translation makes the team problem much harder to solve. In fact, for the BP instance B3 with nine identical agents, the problem is solvable by QDec-FP but QDec-FPS times out. We discussed the reasons for this earlier. On the other hand, problems B11 and B12 were solved by the QDec-FPS planner but were unsolved by QDec-FP. This is most likely due to the many numbers of backtracks required.

We use a boolean variable *SigFlag* that signifies that, when enabled, the original problem description will be preprocessed. Thus the domain will be augmented with all possible signaling macros. To test signaling, we enabled SigFlag for the problems from B13 to B16 in the BP domain. These problems require both signaling and backtracking and cannot be solved by QDec-FP (shown as *na*). For them, we note that QDec-FPS returned no solution when the *SigFlag* was disabled. In the future, we intend to augment QDec-FPS with an ability to automatically turn on *SigFlag* by using an approximate reachability analysis to determine whether signaling is required.

The bottom half of the table shows the results obtained in the Table-Mover (TM) domain. In most problems the agents are non-homogeneous. In contrast to the BP domain, in the TM domain each public action is a collaborative action, e.g., *2lift, 2move, 2drop*. Similar to the BP domain, when backtracking is not required to solve a problem, e.g., the over-simplified instances T1 up to T5, the QDec-FP solver takes less time to find a plan tree than the QDec-FPS solver. Whenever problems are more complex, and backtracking is required, e.g., for the instances T6 and T11, QDec-FPS solves them faster, generally, as it requires fewer backtracks than QDec-FP. There are exceptions, such as instance T10 in which QDec-FPS backtracked 14 times.

The outer loop of Algorithm 1 (Shekhar, Brafman, and Shani, 2019) goes over every possible team solution before it concludes that there is *no solution*. Because there are fewer solutions to the team problem QDec-FPS generates, it is also able to conclude that no solution exists faster than QDec-FP, at times. For example, in instance T9, QDec-FPS backtracked only *six* time before concluding that there is no solution, compared to 31 backtracks for QDec-FP.

We enabled *SigFlag* for the problems T12-T15 in the TableMover domain. Their configuration is simpler than the previous problems, with fewer agents. Problem T13 is interesting because it can be solved without signaling, but it is solved even faster with signaling. There are three agents, φ_1, φ_2, and φ_3 such that φ_1 and φ_3 together are capable to solve this problem without signaling. QDec-FP solved T13 while it backtracked *nine* times. As we purposely placed φ_3 farther from the table's location, QDec-FPS generated a plan where φ_1 signaled to φ_2 the table's location and they achieved the goal together, without φ_3. This solution was generated much quicker and with no backtracks. If we disallow signaling (*SigFlag* is disabled), QDec-FPS still solves this problem with *zero* backtrack, but it takes more time than when *SigFlag* is enabled.

6 Conclusion

In this paper we describe QDec-FPS, an extension of the QDec-FP planner, that solves a multi-agent planning problem by solving a number of single-agent planning problems. The first step is to solve the team problem – where QDec-FPS makes stronger requirements on the plan than QDec-FP by accounting for knowledge of individual agents. This leads to team plans that are more informed, and usually enables faster solution time with fewer backtracks. More importantly, by reasoning about the knowledge of individual agents explicitly, QDec-FPS is able to use signaling – implicit communication through the state of the world, to share information among agents and solve planning problems that were not practically solvable by QDec-FP.

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

References

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 *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*.

Crosby, M.; Jonsson, A.; and Rovatsos, M. 2014. A single-agent approach to multiagent planning. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, 237–242.

Hoffmann, J., and Brafman, R. 2005. Contingent planning via heuristic forward search with implicit belief states. In *Proc. ICAPS*, volume 2005.

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

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

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

Shani, G., and Brafman, R. I. 2011. Replanning in domains with partial information and sensing actions. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, 2021–2026.

Shekhar, S., and Brafman, R. I. 2018. Representing and planning with interacting actions and privacy. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018*, 232–240.

Shekhar, S.; Brafman, R. I.; and Shani, G. 2019. A factored approach to deterministic contingent multi-agent planning. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*, 419–427.