Privacy Preserving Planning in Stochastic Environments

Guy Shani, Roni Stern, and Tommy Hefner
Ben Gurion University of the Negev, SISE Dept., Beer Sheva, Israel

Abstract

Collaborative privacy preserving planning (CPPP) has gained much attention in the past decade. To date, CPPP has focused on domains with deterministic action effects. In this paper, we extend CPPP to domains with stochastic action effects. We show how such environments can be modeled as an MDP. We then focus on the popular Real-Time Dynamic Programming (RTDP) algorithm for computing value functions for MDPs, extending it to the stochastic CPPP setting. We provide two versions of RTDP: a complete version identical to executing centralized RTDP, and an approximate version that sends significantly fewer messages and computes competitive policies in practice. We experiment on domains adapted from the deterministic CPPP literature.

1 Introduction

Designing autonomous agents that act collaboratively is an important goal. A fundamental requirement of such collaboration is to plan for multiple agents acting to achieve a common set of goals. *Collaborative privacy-preserving planning (*CPPP *)* is a multi-agent planning task in which agents need to achieve a common set of goals without revealing certain private information (**?**). In particular, in CPPP an individual agent may have a set of private facts and actions that it does not share with the other agents. CPPP has important motivating examples, such as planning for organizations that outsource some of their tasks.

In this paper we extend the CPPP framework to stochastic domains, where actions may have different effects with varying probabilities. In the planning community, stochastic domains are typically modeled using Markov decision processes (MDPs) (**?**, e.g.). We suggest an MDP formalization for privacy preserving stochastic problems.

A popular approach for solving goal based MDPs is the Real-Time Dynamic Programming (RTDP) algorithm (**?**), computing a value function, estimating the expected cost to the goal from each state. RTDP executes trajectories in the state space, updating the value function along the trajectory, and is guaranteed, under some restrictions, to converge to the optimal value function. We adapt RTDP to our stochastic CPPP setting, showing how agents execute jointly state space trajectories, resulting in the distributed RTDP algorithm (DRTDP).

The complete version of DRTDP follows the same trajectories that will be computed on a centralized MDP. To do so, DRTDP requires a constant synchronization between the agents, which results in many messages. We thus suggest an approximate DRTDP version that requires synchronization only following public actions, which we call the Public Sync RTDP (PS-RTDP). This approximate DRTDP is not guaranteed to converge, and in particular, its trajectories may get into private cycles, requiring restarting the trajectory.

We compare the two versions over a set of domains adapted from the deterministic CPPP literature (**?**), showing that both versions converge to similar expected costs, while PS-RTDP requires as much as an order of magnitude fewer messages.

2 Background

We briefly review background on CPPP, MDPs, and RTDP.

Collaborative Privacy Preserving Planning An MA-STRIPS problem (**?**) is represented by a tuple $\langle P, \{A_i\}_{i=1}^k, I, G\rangle$ where: k is the number of agents, P is a finite set of primitive propositions (facts), A_i is the set of actions agent i can perform, I is the start state, and G is the goal condition.

Privacy-preserving MA-STRIPS extends MA-STRIPS by defining sets of variables and actions as private, known only to a single agent. $private_i(P)$ and $private_i(A_i)$ denote the variables and actions, respectively, that are private to agent i. $public(P)$ is the set of public facts in P. $public_i(A_i)$, the complement of $private_i(A_i)$ w.r.t. A_i, is the set of public actions of agent i. Some preconditions and effects of public actions may be private, and the action obtained by removing these private elements is called its *public projection*, and it is known to other agents. All agents are aware of the execution of a public action, and view the public effects of the action. Goals can be public, or private to one agent.

MAFS (**?**) is a popular algorithm for CPPP — a distributed algorithm in which every agent runs a best-first search to reach the goal. Agents maintain an open list of states, and in every iteration choose a state from the open list to expand, generating all its children and adding them to the open list (avoiding duplicates). Whenever an agent expands a state

that was generated by applying a public action, it also broadcasts this state to all other agents. An agent that receives a state adds it to the open list.

To preserve privacy, the private part of a state is obfuscated when broadcasting it, e.g., by replacing the private facts with some index, such that only the broadcasting agent knows how to map this state index to the corresponding private facts. Once the goal is reached, the agent achieving the goal informs all others, and the search process stops.

Markov Decision Processes In many domains there can be several alternative action effects. First, actions may fail, e.g., in the Logistics domain, loading a package may fail, and the package would remain where it was. Second, actions may have unwanted effects. For example, the driver may enter a wrong destination into the navigation app, and drive to the wrong location. In many cases one can have a stochastic model specifying the probability for each effect to occur.

Such planning domains can be modeled by Markov Decision Processes (MDPs) (**?**). In automated planning, an MDP is a tuple $\langle P, A, I, G, C\rangle$ where P is a finite set of propositions, A is a finite set of actions, I is the start state, G is the goal condition, and C is a cost function. The set of states of the MDP S is the set of all possible assignments to the propositions in P. An action $a \in A$ is defined by preconditions $pre(a)$ as above. However, the action effects $\mathit{eff}(a)$ is a set of stochastic effects $\langle \phi_i, p_i\rangle$, where ϕ_i is a conjunction of positive and negative literals, and p_i is the probability that this specific effect would occur. As some effect must always occur, $\sum_i p_i = 1$. For a pair of states s, s', we denote by $tr(s, a, s')$ the probability that the agent would arrive at state s' by executing a at state s. If $s \models G$, then $tr(s, a, s) = 1, \forall a \in A$. That is, goal states are absorbing. The cost function assigns a cost to executing an action a at a state s. The cost for any action at a goal state $s \models G$ is 0, while executing an action at any other state incurs a positive cost. Hence, the agent strives to arrive at a goal state.

A solution for an MDP is a policy, a mapping $\pi : S \to A$, assigning an action to every state. A policy that minimizes the expected cost for reaching the goal is an optimal solution to an MDP. Many MDP algorithms compute a policy by first computing a value function $V : S \to A$ assigning a value to each state, estimating the expected cost for achieving the goal. The value iteration algorithm computes a value function by iteratively applying the Bellman update:

$$Q(s,a) = C(s,a) + \sum_{s'} tr(s,a,s')V(s') \quad (1)$$

$$V(s) = min_a Q(s,a) \quad (2)$$

over every state until convergence. The resulting policy is: $\pi^V(s) = argmin_a C(s,a)$.

The RTDP algorithm (Algorithm 1) (**?**, e.g.) operates by launching simulated trajectories in state space, starting from the initial state. During a trajectory, at each state s, the algorithm selects a heuristically best action a, and then selects the next state from the $tr(s, a, \cdot)$ distribution. When RTDP reaches a goal state, the trajectory is terminated, and a new one is launched. Bellman updates are used to update the value function along the trajectory. If the value function is initialized optimistically (e.g. to 0), RTDP converges to the optimal value function.

Algorithm 1: RTDP

```
1  RTDP()
2      ∀s ∈ S, a ∈ A, Q(s,a) ← 0, V(s) ← 0
3      while V has not converged do
4          s ← s_0
5          while s ⊭ G do
6              a* ← argmin_a Q(s,a)
7              Q(s,a) = C(s,a) + Σ_{s'} tr(s,a,s')V(s')
8              V(s) ← min_{a'} Q(s,a')
9              sample s' from tr(s,a*,·)
10             s ← s'
```

3 Privacy Preserving MDPs

We extend the definition of CPPP to stochastic domains, modeled as an MDP, with each agent viewing only a part of the complete MDP. More formally, a stochastic collaborative privacy preserving planning problem (SCPPP) is a tuple: $\langle P, \{A_i\}_{i=1}^k, tr, I, G, C\rangle$ where k is the number of agents, P is a finite set of primitive propositions (facts), A_i is the set of actions agent i can perform, I is the start state, G is the goal condition, and C is a cost function. As in a CPPP problem, $private_i(P)$ and $private_i(A_i)$ denote private variables and action of agent i. $public(P)$ is the set of public facts in P, and $public_i(A_i)$ is the set of public actions of agent i. As in an MDP, actions have stochastic effects, i.e. $\mathit{eff}(a) = \{\langle \phi_i, p_i\rangle\}$ where p_i is the probability that effect ϕ_i would occur. As in an MDP, C assigns a cost for executing an action at a state. tr is a transition function, specifying the probability of moving from states s to s' using action a.

While in classical CPPP problems a solution is a sequence of public and private actions, in an SCPPP problem the solution is a policy, determining at each step which agent should execute an action, and which action must be executed. Hence, the policy of an agent i is a mapping $\pi_i : S_i \to A_i \cup \{noop\}$, where S_i is the set of local states — assignments to the private propositions of i and the public propositions. Such a policy either chooses an action to execute at a given state, or lets another agent perform an action.

In this paper we focus on policies that do not allow parallel execution, that is, for each state s of the system, there is exactly one agent where $\pi_i(s_i) \neq noop$, where s_i is the local view of agent i of the global state s. We leave extensions that allow for parallel executions where more than one agent executes an action at each step to future research.

Complete DRTDP We now present our distributed RTDP (DRTDP) algorithm for solving privacy preserving MDP problems. DRTDP operates by running a trajectory that is advanced by one agent at a time. That is, similar to the forward search of MAFS, the trajectory begins with one agent but can then be advanced by other agents. As in MAFS, DRTDP requires message passing between the agents. Like MAFS, messages contain a public state, and indexes of private states. However, as opposed to MAFS, messages also

Algorithm 2: DRTDP for agent i

```
1  DRTDP(i)
2      ∀s ∈ S, a ∈ A, Q_i(s,a) ← 0, V_i(s) ← 0
3      while true do
4          process-messages()
5          if s_c^i ≠ null then
6              advance-trajectory()
7  process-message()
8      foreach Message m = ⟨s,t,v,j⟩ do
9          if m.type = Q-value request then
10             send to m.j ⟨m.s, Q-value response, V_i(s), i⟩
11         if m.type = Q-value response then
12             if m.v < V_i(s) then
13                 V_i(s) = m.v, best_i(s) ← m.j
14         if m.type = trajectory then
15             s_c^i ← m.s
16 advance-trajectory()
17     a* = argmin_a Q_i(s_c^i, a)
18     update(s_c^i, a*)
19     sample s' from tr(s_c^i, a*, ·)
20     s_c^i ←choose-next-agent(s')
21 choose-next-agent(s)
22     if s ⊆ G then
23         broadcast & wait ⟨s_0, Q-value request, ×, i⟩
24         s ← s_0
25     if best(s) = i then return s
26     else send to best(s) ⟨s, trajectory, ×, i⟩, return null
27 update(s, a)
28     foreach s' ∈ tr(s,a,·) do
29         broadcast & wait ⟨s', Q-value request, ×, i⟩
30     Q_i(s,a) = C(s,a) + Σ_{s'} tr(s,a,s') V_i(s')
31     V_i(s) ← min_{a'} Q_i(s,a')
```

contain the current values for the states in the message.

In the complete DRTDP, at each step, a single agent is responsible for advancing the trajectory. This agent requests all other agents to send it their estimated cost to the goal from the current state for their best action. The agent sending the best expected cost is chosen to execute its action and select the next state. This is repeated until the goal is reached. Each agent i maintains the functions $Q_i(s,a)$ and $V_i(s)$.

Algorithm 2 presents a simplified version of our distributed RTDP method. Each agent continuously processes all received messages (line 4). A message m is a tuple $\langle s,t,v,i\rangle$ where s is a state, represented by the public facts and the private state indexes of all agents, v is a (possibly empty) value, i the sending agent, and t is the message type. There can be 3 types of messages: (1) Q-value request: agent i is requesting all agents to send their values for the state s. (2) Q-value response: agent i is answering a Q-value request message with its value for the state s. The receiving agent checks if it has received a better value, and if so, changes $V_i(s)$, also recording the sending agent in $best_i(s)$ (line 13). (3) Trajectory: agent i transfers responsibility of advancing the trajectory from s to the receiving agent.

The private variable s_c^i maintains the current state of the trajectory for agent i, which can be $null$, if the trajectory is currently under the responsibility of another agent. If $s_c^i \neq null$, then i is responsible for advancing the trajectory (line 6). When receiving a trajectory message, s_c^i is set to the received state. When agent i receives responsibility for a trajectory, it first selects its best action a^* (line 17). At this point, we can limit our attention to the actions of agent i, because we already compared the values from all other agents, and i reported the best expected cost.

We then update the Q-function and the value function. The only Q-value that changes is $Q_i(s_c^i, a^*)$ of agent i whose action is executed. Hence, agent i requests (line 29) all other agents to send their values for all possible next states after executing a^* at s_c^i. After all values have been received, we update $Q_i(s_c^i, a^*)$ and $V_i(s_c^i)$ (line 30).

After updating the value function, the agent selects the next possible state s', and then must choose the agent that receives the responsibility to advance the trajectory for s'. First, if s' is a goal state, then the agent must start a new trajectory. To do that, the agent sends a request to all agents for their best value for the initial state s_0. If s' is not a goal state, recall that during the value function update the agent has already requested values from all other agents for all possible next states, including s' (line 29). Hence, agent i already maintains the best agent to handle s' in $best_i(s')$. If the best next agent is i, then it sets s_c^i to s'. Otherwise, it sends a trajectory message to $best_i(s')$, transferring responsibility for the trajectory, and sets s_c^i to $null$. This algorithm results in identical trajectories to the ones generated by RTDP on the joint problem. Thus, the cost added for preserving privacy is only the cost of the message passing mechanism.

Algorithm 3: PS-RTDP for agent i

```
1  advance-trajectory()
2      a* = min_a Q_i(s_c^i, a)
3      sample s' from tr(s_c^i, a*, ·)
4      if private cycle detected then
5          restart trajectory
6      if a* is a private action then
7          local-update(s_c^i, a*), s_c^i ← s'
8      else
9          update(s_c^i, a*), s_c^i ←choose-next-agent(s')
10 local-update(s, a)
11     Q_i(s,a) = C(s,a) + Σ_{s'} tr(s,a,s') min_{a'} Q_i(s',a')
12     V_i(s) ← min_a Q_i(s,a)
```

Public Synchronization RTDP The approximate version of DRTDP, which we call Public Sync RTDP (PS-RTDP), relies on the intuition that often there is no need to interleave the private actions of different agents. That is, as in MAFS, following a private action of agent i, the next action should also be of agent i. Hence, in PS-RTDP, an agent chosen to execute the next action continues to execute additional actions until it executes a public action. Then, the agents vote, as in the complete DRTDP on which agent takes ownership of the trajectory and progresses it farther.

Algorithm 3 shows the differences between the algorithms. In PS-RTDP, the agent responsible for advancing a trajectory considers only its own actions. If the selected action is private, then the agent updates the value using only its own Q-values (line 7), and remains responsible for the tra-

Domain	# Actions	# Facts	Best Cost	Expansions × 10^4	Messages × 10^4	# trajectories + restarts	Total Time (sec)
blocks-2-2	26	26	(4.58 / 4.52)	(0.025 / 0.045)	(0.135 / 0.038)	(40 / 40 + 41)	(0.068 / 0.062)
blocks-3-3	129	63	(7.48 / 7.44)	(0.120 / 0.202)	(1.274 / 0.455)	(60 / 90 + 127)	(0.516 / 0.473)
blocks-4-3	315	99	(9.7 / 9.72)	(0.713 / 0.840)	(7.636 / 2.394)	(80 / 130 + 224)	(4.1 / 3.5)
blocks-5-2	422	107	(11.98 / 12.36)	(5.247 / 5.412)	(30.434 / 9.366)	(150 / 380 + 691)	(35.3 / 29.8)
blocks-6-2	746	146	(14.8 / 16.5)	(68.322 / 68.977)	(399.718 / 117.096)	(900 / 2140 + 7057)	(497.8 / 427.7)
depot-2-3	69	49	(11.2 / 11.4)	(0.276 / 0.973)	(2.844 / 3.219)	(90 / 650 + 62)	(1.2 / 3.0)
depot-3-3	149	74	(13.48 / 13.22)	(2.894 / 3.152)	(29.708 / 8.126)	(220 / 160 + 483)	(22.7 / 15.6)
depot-3-4	334	91	(16.76 / 16.76)	(9.273 / 11.765)	(141.352 / 44.033)	(410 / 240 + 2515)	(79.8 / 55.8)
depot-2-5	245	71	(11.4 / 11.14)	(4.342 / 5.050)	(89.343 / 23.417)	(350 / 170 + 1389)	(55.7 / 23.9)
depot-3-5	407	113	(16.78 / 16.78)	(55.045 / 59.236)	(1136.297 / 289.422)	(2330 / 740 + 10437)	(715.4 / 344.8)
logistics-1-3	49	27	(12.78 / 12.62)	(0.371 / 0.191)	(4.208 / 0.164)	(110 / 40 + 102)	(1.1 / 0.217)
logistics-2-3	69	34	(20.38 / 20.78)	(4.203 / 2.516)	(47.815 / 3.494)	(440 / 140 + 792)	(13.9 / 3.5)
logistics-2-4	80	40	(20.82 / 20.84)	(13.437 / 7.174)	(224.140 / 14.711)	(1130 / 170 + 2440)	(54.4 / 10.9)
logistics-2-5	203	70	(27.0 / 21.58)	(122.934 / 40.523)	(2705.937 / 103.567)	(2134 / 220 + 14162)	(848.5 / 81.7)
logistics-3-4	104	48	(26.8 / 27.22)	(152.786 / 109.649)	(2570.205 / 250.775)	(8010 / 770 + 27091)	(720.6 / 186.0)

Table 1: Empirical comparison of the DRTDP and PS-RTDP. Cell format is DRTDP / PS-RTDP.

jectory. If a^* is a public action, however, the agent uses the value function update mechanism, and selects the next agent to take responsibility as in Algorithm 2 (line 27). As all messages are sent by the global update mechanism and the next agent selection process, reducing calls to these phases reduces the number of messages significantly.

PS-RTDP is no longer guaranteed to converge. One reason is that an agent chosen to advance the trajectory may get stuck in a loop of private actions, without being able to reach the goal, or execute any public action. For example, in the Logistics domain, agent 1 may unload a package at location A. As all agents currently have Q-values of 0, the tie breaking mechanism decided that an agent with no access to A takes the trajectory. That agent has no public actions available, and can hence only advance the trajectory by private driving actions, never releasing responsibility to other agents. We therefore add a private cycle detection mechanism (line 4). If a cycle is detected, we restart the trajectory from s_0. The sensitivity of the cycle detection is important, as cycles may arise due to stochastic effects. Detecting a cycle too soon may make the algorithm ignorant to a possible path towards the goal. Detecting a cycle too late may cause the algorithm to be much slower than the complete DRTDP.

4 Empirical Analysis

We now provide empirical analysis on domains adapted from the CPPP literature, comparing the complete DRTDP and the approximate DRTDP algorithms, implemented in Python. All experiments were run on Google cloud running a Xeon CPU with two 2GHz cores, and 1.8GB user RAM. The domains were adapted by adding stochastic effects. For example, in the blocks world domain, when picking a block it may either be successfully picked, stay where it was, or fall on the table. We also added different capabilities and success probabilities for different agents for completing different tasks, requiring smarter policies for optimization. For example, we added block types, where each arm can lift only a subset of block types, with varying success probabilities.

On each problem we stop the algorithms after 10 RTDP trajectories, and estimate the average cost over 50 policy executions. We terminate once the policy stops improving. Table 1 shows the results. In many domains the approximate DRTDP generated orders of magnitude fewer messages than the complete version, while converging to policies of similar quality. In all domains both methods expanded a similar number of states, but often the approximate version requires fewer goal reaching trajectories, not counting restarts after loops, that can be much shorter.

In the blocks domain, the approximate RTDP generated about one third of the messages. On the other hand, it often required many more trajectories to converge, especially for the larger problems, and hence resulted in only slightly lower runtime. Only in the largest blocks problem, PS-RTDP resulted in significantly higher average cost. This is because in blocks world there is a relatively small number of private actions, and hence the public version is not very different.

In Logistics, the PS-RTDP sends in many cases fewer than $\frac{1}{10}$ of the messages, and is hence considerably faster on many problems. In Logistics, PS-RTDP also required much fewer trajectories to converge. In Depot, as in blocks world, the number of messages sent by PS-RTDP is about one third of the messages sent by the complete DRTDP. In this domain, however, the approximate version required about half the time before converging, especially on larger problems.

5 Conclusion and Future Work

We presented two algorithms for CPPP in stochastic environments. DRTDP is a distributed, privacy preserving adaptation of RTDP, which is identical in execution to RTDP on the joint problem. PS-RTDP is an approximation of DRTDP that shares only public changes between agents to reduce the amount of messages during planning. Experiments show that PS-RTDP sends significantly less messages than DRTDP, sending an order of magnitude fewer messages in some cases. Future research can advance multiple trajectories simultaneously, allowing agents that do not currently advance a trajectory to start a new one. sequential action execution. We will investigate an extension to concurrent execution.

ea fuga nihil illo excepturi similique enim alias obcaecati voluptatum, laborum itaque qui saepe rerum?