Task Scoping for Efficient Planning in Open Worlds - Supplementary Material

Nishanth Kumar*, Michael Fishman*, Natasha Danas,
Michael Littman, Stefanie Tellex, and George Konidaris
Brown University Department of Computer Science
115 Waterman Street, Providence, RI 02912
nishanth_kumar@brown.edu, michaeljfishman@gmail.com
*

In this supplementary material we discuss some details of our algorithm, its constraints, an example, a theory for generalization to multi-valued state-based reward functions, some preliminary results, and a brief description of current and future work.

Algorithm

Please see the algorithm described in our Extended Abstract and the example in this supplementary material for variable definitions.

Our algorithm represents the transition dynamics of a domain in terms of (precondition, action_name, effect) triples, where each precondition is the weakest precondition under which action_name has the specified effect. We use this representation because the agent only needs to care about the portions of preconditions that influence effects on variables of interest. However, this representation is problematic when there are preconditions which have components such that there is a different effect depending on whether a component is true or false, and where each of these effects includes variables of interest. This issue did not come up in our evaluation. An example follows.

Imagine the taxi domain from Dietterich (2000), modified to that $Move_N$ is defined by ($\neg blinker\_on$,$Move\_N$,$taxi.y + +$), ($blinker\_on$,$Move\_N$,$taxi.y + +, taxi.x + +$). Since Process_Conditions represents (precondition,action_name,effect) triples using weakest preconditions, it would instead represent $Move\_N$ as $(True, Move_N, taxi.y{+}{+})$, $(blinker\_on, Move_N, x{+}{+})$. From this representation, it would not consider $\neg blinker\_on$ as a precondition, and so if $blinker\_on$ were true in the start condition, the algorithm would add it to causal_links and scope it out. This would make the optimal policy impossible to describe if, for example, the passenger and passenger's goal were both due North of the taxi, so that moving diagonally North-East would be less efficient than turning the blinker off and heading North. We believe we can fix this constraint by using tools from the Formal Methods community (specifically geometric logic).

*Corresponding authors equally contributed; others co-advised.

Example

As an example, consider a multi-passenger variant of the popular Taxi domain from Dietterich (2000). The agent controls a taxi that can only fit 1 passenger at a time and the domain contains 1 relevant passenger, and 9 irrelevant passengers. Let p refer to a generic passenger. The state variables, actions, and reward are:

- $\forall\, p$: $p.x$, $p.y$, $p.x_g$, $p.y_g$, $p.inTaxi$, $p.relevant$
- For a singleton taxi: $taxi.x$, $taxi.y$
- Movement actions: $Move_N$, $Move_S$, $Move_E$, $Move_W$
- Passenger actions $\forall p$: $Pickup(p), Dropoff(p)$
- Rewarded for dropping off relevant ps, not irrelevants:

$$(p.relevant) \vee (p.x = p.x_g \wedge p.y = p.y_g \wedge \neg p.inTaxi)$$

Where $p.x_g, p.y_g$ refer to goal coordinates of passenger p.

The scoping algorithm operates on the multi-passenger taxi domain as follows:

1. The reward clause for each irrelevant passenger is always false, so the agent need only consider the relevant passengers. Call the relevant passenger p_0.
2. In the start state, $p.inTaxi$ is false for any p, so the agent only addresses $p_0.x = p_0.x_g$ and $p_0.y = p_0.y_g$.
3. The only actions that affect these variables are the movement actions, so the agent considers these to be relevant.
4. The precondition for moving p_0 is $p_0.inTaxi = True$.
5. The only action that affects this condition is $Pickup(p_0)$.
6. The precondition for $Pickup(p_0)$ is

$$p_0.x = taxi_x \wedge p_0.y = taxi_y \wedge \forall p : p.inTaxi = False$$

7. The colocation conditions are affected by the taxi's movement actions, which have no precondition.
8. The $Pickup(p_0)$ condition threatens the causal-link (start-condition, $p_0.inTaxi = False$, goal-condition), so we now must consider all actions that affect $p_0.inTaxi$.
9. The only actions that affects $p_0.inTaxi$ are $Pickup(p_0)$, which has already been considered, and $Dropoff(p_0)$, which is now marked as relevant.

10. Each mentioned subgoal is now either guaranteed by the start condition and set of relevant actions, or has all affecting actions in the set of relevant actions. The agent concludes that the relevant state variables are those mentioned in the non-guaranteed subgoals, and the relevant actions are those used to satisfy the subgoals. Thus, all irrelevant passengers can be removed to massively decrease the state-space.

Guarantees

Equisolvability 1. *Assuming the constraint mentioned in the Algorithm section, the scoping algorithm returns a domain that is solvable iff the original domain is solvable, and whose optimal policy is an optimal policy for the original domain*

Proof. As can be seen from see the algorithm described in our Extended Abstract, the agenda variable is populated by the goal conditions and the preconditions discovered by Process_Conditions. Each condition that is discovered will be put into causal_links if it is implied by the start condition. Find_Threats moves from causal_links to agenda any condition which contains variables that may be affected by any of the used_actions. A condition cannot be made false if no variable in it is changed, so Find_Threats ensures that, when Scope_Task terminates, all conditions in causal_links remain true throughout any trajectory with the specified start condition and actions. By the constraint mentioned in the Algorithm section, the only conditions the agent may want to be false will appear explicitly negated in the precondition of a (precondition, action, effect) triple. Thus, when Scope_Task terminates, there are no conditions the agent may need to be false contained in causal_links.

Process_Conditions ensures that any goal or precondition not guaranteed to remain true throughout the trajectory has all actions that affect any of its variables added to used_actions, and all preconditions of those actions added to agenda. By induction, any condition that can affect future reward, assuming the specified starting condition and used_actions, is guaranteed to remain true throughout the trajectory, has all affecting actions included in used_actions, or has its effect on future rewards mediated entirely by conditions that are guaranteed to remain true throughout the trajectory.

Relevant_vars is the set of all discovered variables that are not guaranteed to have the correct value throughout the trajectory, so relevant_vars and used_actions are guaranteed to be a sufficient state and action space for optimal planning. □

Multi Valued State-Based Reward

Here we describe our handling of multi-valued state-based reward. We will split the reward-mentioned state variables into two categories:

State variables that occur only in conditions in the reward function: For each condition mentioned in the reward, call the condition *dominant* if, for each assignment of values to all variables not mentioned in the condition, the reward is never lower when the condition is true, and for at least one assignment of variables, the reward is higher. Then the dominant conditions can be treated like goal conditions by the scoping algorithm.

State variables that occur outside of conditions in the reward function: In this work, we do not prune any state variables mentioned in the reward function outside of conditions. We believe techniques for pruning these variables exist, but are out of scope of this paper.

Preliminary Experiments and Results

To test our approach, we implemented a version of our task scoping algorithm and ran it on various FMDP's expressed in a small subset of Sanner (2010)'s RDDL language. We ran various state-of-the-art FMDP planners from the Probabalistic Track of the 2018 International Planning Competition (IPC 2018) on some domains before and after having performed Task Scoping and compared their performance. Our method heavily used the pyrddl library to parse the rddl files (Bueno 2018 2019). Our open-source code is available on GitHub at: `https://github.com/h2r/OO-Scoping-IPC/tree/scoping`.

We implemented various instances of the multi-passenger taxi domain (pictured in Figure 1) with different numbers of grid-sizes, relevant and irrelevant passengers. In all cases, our scoping algorithm was able to correctly parse the domains and recognize that the irrelevant passengers were not necessary to achieving the goal.

We primarily tested two variants of the PROST planner (Keller and Eyerich (2013) and Geisser and Speck (2018)) and the SOGBOFA planner (Cui, Keller, and Khardon (2019)) which were the top-performing FMDP planners as of the IPC 2018. We ran these algorithms on progressively larger instances of the multi-passenger taxi domain to discover how they scaled with the size of the state space and number of irrelevant variables present. As expected, all these planners scaled roughly exponentially with the total size of the state space. On a domain with a 6×6 grid (36 possible grid states), 28 irrelevant passengers and 1 relevant passenger, all versions of the PROST and SOGBOFA planners failed to find any solutions to the domain when given a planning horizon of 21 or lesser (the optimal policy would only require a horizon of 18).

Our task-scoping algorithm was able to remove all 28 irrelevant passengers from this 6×6 grid domain in less than 30 seconds. By contrast, both the PROST and SOGBOFA planners took much longer to pre-compute their data structures before they could begin planning. Furthermore, after having successfully scoped this domain, the PROST planners became able to solve it and obtain the optimal policy with the same planning horizons.

Conclusion and Future Work

We presented ongoing work on developing a novel abstraction algorithm for large, open-world problems expressed as FMDP's. Our Task Scoping algorithm leverages knowledge of an agent's initial state to prune irrelevant variables from the FMDP in such a manner as to provably preserve

the optimal value function. Preliminary experiments indicate on very-large variants of the multi-passenger taxi domain indicate that our method is promising and renders previously-intractable problems tractable to current state-of-the-art FMDP planners.

In the future, we first hope to test our approach more exhaustively. The primary bottleneck thus far has been that our current implementation is able to work on only a small subset of the RDDL language. Thus, we have only been able to experiment with domains that are expressable in this limited RDDL vocabulary. However, this is primarily an engineering challenge since our algorithm and method are able to generalize to almost any FMDP expressable with RDDL and even PPDDL Younes and Littman (). We hope to expand our implementation and thus test our algorithm on many more domains, such as those from the IPC itself. Additionally, we hope to demonstrate our algorithm's ability to handle truly 'open-world' problems by solving various tasks within very large domains such as open-world video games expressed as FMDP's.

Additionally, we hope to generalize our algorithm to handle arbitrary rewards more effectively and even partially-observable domains. Such an extension would allow us to utilize our task scoping method to enable robots to plan efficiently in real-world environments, which could allow robots to perform a plethora of tasks that are currently thought impossible.

References

[2018 2019] Bueno, T. 2018-2019. pyrddl. `https://github.com/thiagopbueno/pyrddl`.

[2019] Cui, H.; Keller, T.; and Khardon, R., eds. 2019. *Stochastic Planning with Lifted Symbolic Trajectory Optimization*. AAAI Press.

[2000] Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Int. Res.* 13(1):227–303.

[2018] Geisser, F., and Speck, D. 2018. Prost-dd-utilizing symbolic classical planning in thts. *Sixth International Probabilistic Planning Competition (IPPC-6): planner abstracts*.

[2013] Keller, T., and Eyerich, P. 2013. Prost: Probabilistic planning based on uct. In *Proceedings of the Twenty-Second International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'12, 119–127. AAAI Press.

[2010] Sanner, S. 2010. Relational dynamic influence diagram language (rddl): Language description. http://users.cecs.anu.edu.au/ ssanner/IPPC$_2$011/$RDDL.pdf$.

[] Younes, H. L. S., and Littman, M. L. Ppddl 1.0: An extension to pddl for expressing planning domains with probabilistic effects. Technical report.