Planning with Partially Specified Behaviors

Javier SEGOVIA-AGUAS [a] Jonathan FERRER-MESTRES [a] Anders JONSSON [a]
[a] *DTIC, Universitat Pompeu Fabra, Spain*

Abstract. In this paper we present a framework called PPSB for combining reinforcement learning and planning to solve sequential decision problems. Our aim is to show that reinforcement learning and planning complement each other well, in that each can take advantage of the strengths of the other. PPSB uses partial action specifications to decompose sequential decision problems into tasks that serve as an interface between reinforcement learning and planning. On the bottom level, we use reinforcement learning to compute policies for achieving each individual task. On the top level, we use planning to produce a sequence of tasks that achieves an overall goal. Experiments show that our framework is competitive with realistic environments where a robot has to perform some tasks.

Keywords. Agent Programming, Hierachical Decomposition, Classical Planning, Reinforcement Learning

1. Introduction

Reinforcement learning (RL) and planning are two popular techniques for solving sequential decision problems, i.e. problems in which an agent has to make repeated decisions in order to achieve some objective. Although the two techniques are similar in many ways, their relationship has not often been studied in the literature. In this paper we compare and contrast RL and planning, with the goal of proposing a framework that takes advantage of the strengths of each technique.

In planning, the state is represented by variables (in RL learning this is known as a *factored* representation). Furthermore, planning requires a *model* of each action that describes its applicability and effects. Finally, in planning the aim is to reach a goal, at which point the decision problem ends. Partially as a result of the International Planning Competition, there exists a range of domain-independent planners that can efficiently solve a variety of large planning problems. Although one can represent stochastic effects and partial observability in planning, state-of-the-art planners perform better when actions have deterministic effects and the problem is fully observable.

On the other hand, in RL the aim is to learn a policy (i.e. a mapping from states to actions) that maximizes some measure of expected future reward, which includes but is not limited to reaching a goal. So called *model-free* RL techniques can learn a policy from experience even in the absence of action models. RL is also equipped to handle stochastic actions. However, most RL techniques are slow and take a long time to converge to an optimal policy.

We propose a framework that we call Planning with Partially Specified Behaviors, or PPSB based in the *PLANQ-learning* method [14]. We assume that a domain expert provides a partial specification of high-level actions, including action models, but that no model is available for low-level actions. Given such a partial specification, PPSB defines a set of behaviors or tasks that serve as an interface between planning and RL. On the bottom level, we use RL to learn the policy of each individual task. This setting suits RL: no action models are available, actions are stochastic and the scope of each individual task is limited. On the top level, we use planning to compute a sequence of tasks that achieves an overall goal. Although individual actions are stochastic, tasks are deterministic from the perspective of planning, making it possible to solve the problem efficiently.

The main limitation of PPSB is the assumption that a domain expert is at hand to provide a partial specification. We believe, however, that this assumption is not unrealistic. In many real-world tasks we can use discrete high-level variables to describe progress towards the goal. For example, we can model the problem of a robot delivering coffee to a human using binary variables such as "the robot is holding a cup", "the cup contains coffee", etc. The effect of high-level actions on such variables is also relatively easy to model. In contrast, the effect of low-level actions such as motion actuators are more difficult for a domain expert to model.

The rest of the paper is organized as follows. We formally describe planning and reinforcement learning in Sections 2 and 3, respectively. In Section 4 we present PPSB and define the precise conditions that have to hold in order to apply PPSB. We then present results from experiments with PPSB in Section 5. In Section 6 we describe related work and we conclude with a discussion in Section 7.

2. Planning

The idea in planning [15,16,6] is to find a sequence of actions or policies that produces a desired goal state. The advantage of planning is that there exists a wide range of domain-independent planners that can efficiently solve large planning instances for a variety of domains. The main limitation is that these planners require a model of the actions, which can be difficult to obtain in realistic problems. Furthermore, planning works best when actions have deterministic effects, and the performance tends to degrade in the face of uncertainty and partial observability.

In this paper we consider the STRIPS fragment of planning [5]. A STRIPS planning problem is a tuple $\mathscr{P} = \langle F, A, I, G \rangle$, where

- F is a set of propositional variables or fluents,
- A is a set of actions, and each action $a \in A$ has a precondition $\mathsf{pre}(a) \subseteq F$, an add effect $\mathsf{add}(a) \subseteq F$ and a delete effect $\mathsf{del}(a) \subseteq F$,
- $I \subseteq F$ is the initial state given by a subset of fluents,
- $G \subseteq F$ is a goal state given by a subset of fluents.

A STRIPS planning problem $\mathscr{P}$ implicitly defines a state space $S = 2^F$, and a state $s \subseteq F$ is defined as a subset of fluents that are true in s, while fluents in $F \setminus s$ are assumed to be false. The set of goal states is $S_G = \{s \subseteq F \mid G \subseteq s\}$, i.e. states in which all fluents in the goal G are true. An action $a \in A$ is applicable in state $s \in S$ if and only if $\mathsf{pre}(a) \subseteq s$, and applying a in s results in a new state $\theta(s,a) = (s \setminus \mathsf{del}(a)) \cup \mathsf{add}(a)$.

The aim of planning is to compute a plan, i.e. a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ that induces a state sequence $s_0, s_1, \ldots, s_n$ with $s_0 = I$ such that for each i, $1 \leq i \leq n$, a_i is applicable in s_{i-1} and $s_i = \theta(s_{i-1}, a_i)$ is the result of applying a_i in s_{i-1}. The plan π *solves* planning problem $\mathscr{P}$ if and only if s_n is a goal state, i.e. if $G \subseteq s_n$.

3. Reinforcement Learning

Reinforcement Learning (RL) [12] is a family of algorithms that aim at computing a policy, i.e. a mapping from states to actions, that maximizes some measure of expected future reward. Most RL algorithms are value-based, maintaining and updating a value function that implicitly defines a policy. Model-free RL algorithms can learn an optimal or near-optimal policy even in the absence of action models, i.e. even when the effect of actions is completely unknown and stochastic. The drawback of RL is that it often struggles to reach a goal for the first time, and convergence tends to be slow for large problems.

3.1. Markov Decision Process

An RL problem is usually modelled as Markov decision process, or MDP, a fully observable stochastic representation of a sequential decision problem. Formally, an MDP is described by a 4-tuple $\mathscr{M} = \langle S, A, P, R \rangle$ where S is a set of states, A is a set of actions, $P : S \times A \times S \to [0,1]$ is a transition probability function satisfying $\sum_{s' \in S} P(s, a, s') = 1$ for each state action pair $(s, a) \in S \times A$, and $R : S \times A \to \mathbb{R}$ is an expected reward function. Given a current state $s_t \in S$, action $a_t \in A$ can be applied following a policy $a_t = \pi(s_t)$ that results in a scalar reward $r_t \sim R(s, a)$ and the system transitions to state $s_{t+1} \in S$ with probability $P(s_t, a_t, s_{t+1})$. MDPs are expressive enough to model both continuing and episodic decision problems; in the former, execution continues indefinitely, while in the latter, there is a set of terminal states $S^+ \subseteq S$ and execution stops when a state in S^+ is reached. The aim of an MDP is to learn a policy $\pi : S \to A$, i.e. a mapping from states to actions, that maximizes the expected future reward $\mathbb{E}(R_t) = \mathbb{E}(\sum_{i=0}^{\infty} \gamma^i r_{t+i})$. Here, $\gamma \in (0, 1]$ is a discount factor that bounds the expected future reward. For episodic tasks, one can often choose $\gamma = 1$.

3.2. Q-Learning

Q-learning is a common RL algorithm for solving an MDP $\mathscr{M}$. The algorithm maintains and updates an action value $Q(s, a)$ for each state-action pair $(s, a) \in S \times A$. The action value $Q(s, a)$ estimates the expected future reward when choosing action $a \in A$ in state $s \in S$. Together, the action values implicitly define a policy as $\pi(s) = \arg\max_{a \in A} Q(s, a)$. After taking action a_t in state s_t, observing a reward r_t and transitioning to a new state s_{t+1}, the algorithm updates the action value for (s_t, a_t) as

$$Q(s_t, a_t) \leftarrow (1 - \alpha) Q(s_t, a_t) + \alpha [r_t + \gamma \max_{a' \in A} Q(s_{t+1}, a')],$$

where $\alpha \in (0, 1)$ is a learning rate. When action values are stored in a lookup table and the value of α asymptotically approaches 0, Q-learning is guaranteed to converge to an optimal policy even when the transition probability function P and expected reward function R are unknown.

4. Planning with Partially Specified Behaviors

In this section we describe Planning with Partially Specified Behaviors (PPSB), our framework for combining planning and reinforcement learning. As previously mentioned, PPSB as *PLANQ-learning* method decomposes a sequential decision problem into a set of tasks and uses reinforcement learning to learn the policy of each individual task. At the top level, PPSB uses classical planning to compute a sequence of tasks that achieves an overall goal. In this paper we take the somewhat extreme view that tasks are deterministic from the perspective of planning: when performing a task, even though individual actions are stochastic, the task policy keeps executing until the task objective is met. The environments we are working with have no deadends. Thus, each task can be trained independently, but still is better to train them following the plan sequence. Then the low-level state is guided to match each task precondition in a specific order speeding up the learning phase. To achieve this, we make several assumptions that are explained below. In Section 7 we discuss how some of these assumptions can be relaxed.

PPSB assumes that the state is partially factored, and that there is a known set of high-level variables V_h, such that each variable $v \in V_h$ has finite domain $D(v)$. The state space S is composed of a low-level and a high-level part, i.e. $S = S_l \times S_h$, where $S_h = \times_{v \in V_h} D(v)$ is the joint domain of the variables in V_h. The composition of the low-level state space S_l can be arbitrary. PPSB models episodic tasks, and the aim is to reach a goal, which is expressed as an assignment of values to the high-level variables in V_h.

Likewise, PPSB assumes that the action set A is composed of low-level and high-level actions, i.e. $A = A_l \cup A_h$. Low-level actions in A_l only change the low-level part of the state (i.e. S_l) but their effect may depend on the high-level part of the state (i.e. S_h). In contrast, high-level actions in A_h may change both the low-level and high-level part of the state. PPSB does not require a model of the low-level actions. In contrast, PPSB assumes that each high-level action $a \in A_h$ has a precondition $\mathsf{pre}(a)$ and an effect $\mathsf{eff}(a)$, both partial assignments of values to the high-level variables in V_h. In addition, the precondition $\mathsf{pre}(a)$ specifies a set of low-level states $S_l^a \subseteq S_l$ in which a is applicable. The effect of a can be stochastic, but for tasks to be deterministic, a is *not* allowed to change the values of high-level variables that do not appear in $\mathsf{eff}(a)$. The only two possible outcomes of a with respect to high-level variables is to achieve its effect $\mathsf{eff}(a)$ or leave all values unchanged and continue executing the policy in contrast to *PLANQ-learning* that stops and replan if the goal is not met. The effect of a on low-level variables can be arbitrary and is not modelled. Given the above setting, PPSB defines a set of tasks T with one task $\tau \in T$ per high-level action $a \in A_h$. For each task $\tau \in T$ we define an MDP $\mathscr{M}_\tau$ and use Q-learning to learn a policy that estimates a solution to $\mathscr{M}_\tau$. At the top level we define a planning problem $\mathscr{P} = \langle F, T, I, G \rangle$ with the tasks of T as actions. Below we describe how to construct $\mathscr{M}_\tau$, $\tau \in T$, and $\mathscr{P}$.

4.1. Task MDPs

Let $\tau \in T$ be the task associated with high-level action $a \in A_h$. The MDP $\mathscr{M}_\tau$ is defined as $\mathscr{M}_\tau = \langle S, A_l, P, R_\tau \rangle$, i.e. $\mathscr{M}_\tau$ includes all states in $S = S_l \times S_h$ but only low-level actions in A_l. The task MDP $\mathscr{M}_\tau$ is episodic, and the set of terminal states equal $S_\tau^+ = \{(s_l, s_h) \in S_l \times S_h \mid s_l \in S_l^a\}$, i.e. all states that satisfy the precondition $\mathsf{pre}(a)$ of a with respect to the low-level part of the state. The transition probability function P is unknown, while we define an expected reward function R_τ that returns a uniform reward of -1 everywhere. This way, the optimal policy is to reach a terminal state as quickly as possible. Note that $\mathscr{M}_\tau$ does not include the high-level action a itself. Instead, to perform a task $\tau \in T$ PPSB first executes a policy for $\mathscr{M}_\tau$ until a terminal state is reached, and then separately applies a, possibly repeating the whole process in case a does not achieve its effect on the high-level variables until it reaches a terminal state again to apply a. Since only low-level actions in A_l are part of $\mathscr{M}_\tau$, a policy for $\mathscr{M}_\tau$ can never change the high-level part of the state (the reason S_h is part of the state is that low-level actions may have different effects for different high-level states).

4.2. High-Level Planning Problem

In this section we describe how to construct the high-level planning problem $\mathscr{P} = \langle F, T, I, G \rangle$. The fluents in F model the values of the high-level variables in V_h, but since fluents are binary, we include one fluent per variable-value pair, i.e. $F = \{v[d] \mid v \in V_h, d \in D(v)\}$. The initial state I is constructed from the current value of the high-level variables, while the goal state G is constructed from the goal on the high-level variables. The set of tasks T corresponds to the set of high-level actions A_h, and each task $\tau \in T$ (with associated high-level action $a \in A_h$) has precondition $\mathsf{pre}(\tau)$, add effect $\mathsf{add}(\tau)$ and delete effect $\mathsf{del}(\tau)$, each derived from the precondition $\mathsf{pre}(a)$ and effect $\mathsf{eff}(a)$ of a. Specifically, the precondition equals $\mathsf{pre}(\tau) = \{v[d] \in F \mid \mathsf{pre}(a)(v) = d\}$, i.e. the set of fluents modelling values assigned to variables by $\mathsf{pre}(a)$. The add and delete effects of τ are similarly defined.

4.3. PPSB Algorithm

In this section we describe the global PPSB algorithm. As described above, PPSB takes as input a partial specification of high-level actions, and decomposes the overall sequential decision problems into tasks, one per high-level action. PPSB then computes an overall plan (i.e. sequence of tasks) for reaching the overall goal, and executes this task sequence in order to achieve the goal, using Q-learning to learn the policy of each individual task.

Algorithm 1 shows the pseudo-code of the PPSB algorithm. To ensure that the policy of each individual task is properly learned, the loop on line 6 executes the plan π many times, possibly for different initial low-level states. An alternative would be to consider different initial high-level states and recompute the plan π in each iteration. Executing the plan π consists in applying each task τ_i of π in sequence. The loop on line 9 is necessary to ensure that the effect of the associated action a_i has been achieved, which is a necessary requirement to complete task τ_i.

5. Experiments

We tested the PPSB algorithm extensively in the Tiding-Up domain, a complex domain in which a robot has to place objects in their corresponding room. The world is organized as follows: There is a robot with a gripper, a set of objects, a set of rooms and a set of buttons where each button opens one room and close the other ones. The idea is to learn different skills (in our case behaviors) for each possible interaction of the robot

Algorithm 1: PPSB Algorithm

input : Partial specification V_h, S_l, A_h, A_l, goal on V_h.
output: Q-values for each task policy, $Q_\tau(s,a)$

Define a set of tasks T from the high-level actions in A_h;
Construct a task MDP $\mathscr{M}_\tau$ for each task $\tau \in T$;
Initialize Q-values $Q_\tau(s,a)$ for each task $\tau \in T$;
Construct high-level planning problem $\mathscr{P} = \langle F, T, I, G \rangle$;
Compute a plan $\pi = \langle \tau_1, \ldots, \tau_n \rangle$ that solves $\mathscr{P}$;
while *there are simulations remaining* **do**
  Initialize high-level variables to I;
  for *each task $\tau_i \in \pi$ with associated action $a_i \in A_h$* **do**
    while *the effect of a_i has not been achieved* **do**
      Execute the policy for $\mathscr{M}_{\tau_i}$ until termination, simultaneously updating the Q-values for τ_i;
      Apply action a_i;
    end
  end
end

with the objects and the buttons. Figure 1 shows an example instance of the Tiding-Up domain. The robot can only observe objects in a small radius around itself, the buttons can be observed when the robot is on top of them, and rooms when it is inside them; the resulting model is in fact partially observable, but is well approximated by a MDP. We used Q-Learning of the RL tasks. We ran the experiments with objects, buttons and rooms intentionally placed far from each other to make the problem harder. To run the experiments we used the Gazebo simulator [19]. Actions given by RL are applied by the robot on the simulated world and the resulting state is given by the simulator itself. The experiments were performed on an Intel Core i5-4590 CPU running at 3.3 Ghz x 4 with 8 GB of RAM.

The state is encoded with 8 variables with variables $v_1 - v_5$ encoding the high-level state and $v_6 - v_8$ encoding the low-level state.

- v_1: Whether or not button 1 is pressed
- v_2: Whether or not button 2 is pressed
- v_3: Whether or not object 1 is in room 1
- v_4: Whether or not object 2 is in room 2
- v_5: Whether or not the gripper is holding an object
- v_6: The X robot position
- v_7: The Y robot position
- v_8: The direction of the robot

The domain of variables $v_1 - v_5$ are all binary while v_6 and v_7 depend on the discretization of the environment, which for this case is 80. While the possible directions of the robot v_8 are 8. Thus, the total size of the state space is $|S| = 2^5 \times 80^2 \times 8 = 1,638,400$.

The low-level actions in A_l are all actions for moving the robot forwards, or turn clockwise or counter-clockwise. These actions only modify the low-level variables $v_6 - v_8$. The high level actions A_h are specified as follows using $X \in [1,2]$:

- "Step On Button X"
  - ∗ pre: v_X = false (button X not pressed)
  - ∗ eff: v_X = true (button X pressed)
  - ∗ terminal state: robot is on button X
- "Grasp Object X"
  - ∗ pre: v_5 = false (not holding an object)
  - ∗ eff: v_5 = true (holding an object)
  - ∗ terminal state: robot can grasp object X
- "Place Object X in room Y"
  - ∗ pre: v_5 = true (holding an object)
  - ∗ eff: v_{X+2} = true (object in room Y), v_5 = false (not holding an object)
  - ∗ terminal state: robot is in room Y holding object X

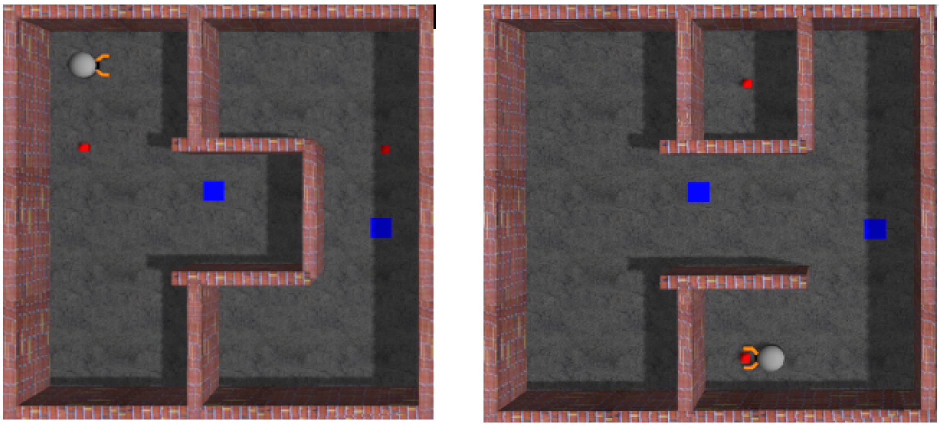

Figure 1. The initial and goal state

Initial and goal state of Tiding-Up domain. Small red boxes have to be placed in the top and bottom rooms, respectively. Big blue squares are the buttons which open a corresponding room and close the other one.

Figure 2 shows the monotonically decreasing average running steps along the simulation of the gripper in the grid. The average starts with hundreds of steps and the robot learns how to perform every task with less steps as more simulations are executed. The simulation has a bound of $10,000$ episodes. However, after 100 episodes the values converge.

6. Related Work

Our work is highly related to other hierarchical approaches to reinforcement learning and planning. In RL, there exist three main approaches to task decomposition: options [13],

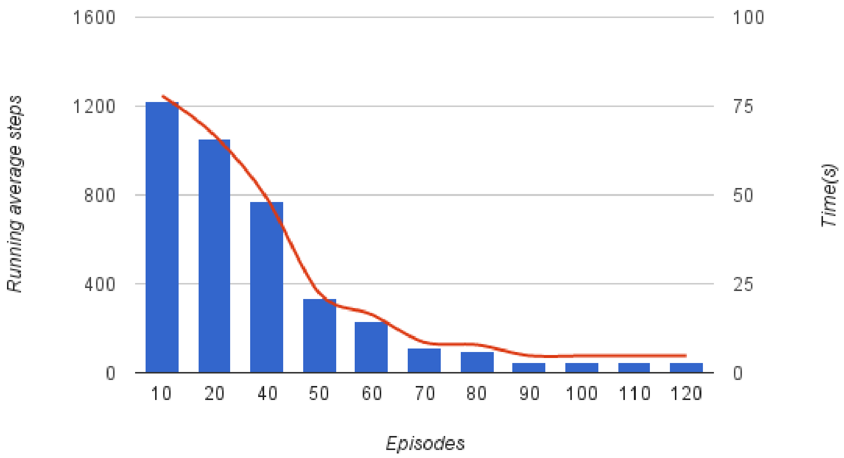

Figure 2. The performance of PPSB on a 24×24 discretized grid.

MAXQ [3] and HAMs [11]. In planning, Hierarchical Task Networks, or HTNs [4], are a popular framework for task decomposition.

The options framework is a representation of semi-Markov decision processes in which options are treated as primitive actions. A policy is then learned directly over options, and the framework makes it possible to mix options and primitive actions. This is the approach we use in our HRL implementation. In MAXQ, which is a more integrated approach, the action-values of individual tasks are taken into account when updating the policy over tasks. An associated algorithm, MAXQ-Q, performs Q-learning to learn the policies of the individual tasks as well as the high-level task. Then, MAXQ-Q high-level policies are affected by the low-level policies in spite of PPSB, but PPSB has the chance to modify planning actions costs with the low-level policy values. In this case, PPSB could improve the plan and policies quality by performing a cycle of learning low-level policies, updating planning action cost and replanning every few iterations. Finally, in HAMs, finite state machines are used to represent the task decomposition. PPSB is different from these frameworks in that it uses planning instead of reinforcement learning to compute a high-level plan.

HTNs, on the other hand, do not only require a task decomposition, but also a partial order on the subtasks to be performed, else planning becomes inefficient. In addition, just like other planning approaches, it requires a model of the low-level actions. In contrast, PPSB does not require a model of the low-level actions nor a partial order on the subtasks; it uses planning to decide the order among the subtasks and model-free RL techniques to learn individual task policies.

Also there is a pure planning decomposition with combined task and motion planning [20] and for mobile manipulation [21]. The first one uses symbolic planning in a hierarchical approach and for each action performs a geometric reasoning to a goal of the top level action. While the second one uses state abstraction and HTNs mixed in an algorithm called SAHTN to guide the search for the motion planner.

Perhaps the two approaches most related to ours are that of [9] and [14]. The first one uses a planning algorithm to derive a task decomposition in the form of a HAM, i.e. a finite-state machine. Their approach is different from ours in that they use a linear temporal logic (LTL) specification to synthesize a finite state machine that encodes a set of possible solutions to the high-level task. RL is then used at the choice points to learn a policy over the high-level tasks, restricted by the finite state machine. The second one is a task decomposition that uses symbolic planning (STRIPS) in the top and performs RL in the bottom level but they do not deal with the uncertainty of the low level, so if some low level action affects to the preconditions of the high level and it does not match the goal they replan from the current state. In our case, only high level actions can change the high level part of the state while low level actions follow policies to terminal states for each individual task.

Although there are other recent approaches regarding how high-level actions rules could be learned. There is a new work to obtain the model with costs [22] using the system *NLOCM*. This system is an extension of a set of algorithms called *LOCM* including numeric weigths to its finite state automata model and using constraint programming to learn. Another work is a framwork to learn the model in environments with uncertainty [23]. They start by generating the operator candidates that may describe the model, solving which are the best among those candidates using planning techniques.

7. Discussion

In this paper we have presented a framework called PPSB based on *PLANQ-learning* that combines planning and reinforcement learning to take advantage of the strengths of each technique. Planning is used to compute a high-level solution in the form of a task sequence, and reinforcement learning is used to learn a policy of each individual task.

There are many possible research directions in which to continue this work. First note that PPSB is flexible in terms of the exact planning algorithm used at the top level and the RL technique used to learn the policies of individual tasks. Potentially, we can modify the framework to incorporate other elements into the planning problem and the task MDPs, e.g. continuous states and actions at the low level. Another possibility is to use Iterative Width planners [17] to search in the low level exploiting the structure of the state instead of learning a policy.

If we relax the assumption that high-level actions have a single possible effect on the high-level variables, the associated tasks of the high-level planning problem are no longer deterministic. In this case we would have to use a different planning technique such as a Full Observable Non-Deterministic (FOND) planner [18,10] to solve the high-level problem in the face of non-determinism. Although such a planner does not scale as well as a deterministic planner to large problems, it should still be able to solve the high-level planning problem when the number of high-level actions is not too high.

Another issue related to the scalability of the approach as the number of high-level actions grows is the increasing number of task MPDs to be solved. In planning it is common to *parameterize* actions, and the same idea might work in PPSB: defining high-level actions that are parameterized on some variable, and using RL to learn a generalized policy for the family of associated tasks. The concept of *skills* may be useful in this context [8]. It is also common to perform *state abstraction* when learning the policy for tasks in a hierarchical RL setting [1,2,7], and the same idea could be applied in PPSB.

References

[1] D. Andre and S. Russell. State Abstraction for Programmable Reinforcement Learning Agents. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-02)*, pages 119–125, 2002.

[2] T. Dietterich. State Abstraction in MAXQ Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems 12*, pages 994–1000, 1999.

[3] T. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[4] K. Erol, J. Hendler, and D. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 1123–1128, 1994.

[5] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.

[6] H. Geffner and B. Bonet. A Concise Introduction to Models and Methods for Automated Planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(1):1–141, 2013.

[7] A. Jonsson and A. Barto. Automated State Abstraction for Options using the U-Tree Algorithm. In *Advances in Neural Information Processing Systems 14*, pages 1054–1060, 2001.

[8] G. Konidaris and A. Barto. Building Portable Options: Skill Transfer in Reinforcement Learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 895–900, 2007.

[9] M. Leonetti, L. Iocchi, and F. Patrizi. Automatic generation and learning of finite-state controllers. In *Artificial Intelligence: Methodology, Systems, and Applications*, pages 135–144. Springer, 2012.

[10] C. Muise, S. McIlraith, and C. Beck. Improved Non-Deterministic Planning by Exploiting State Relevance. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS-12)*, 2012.

[11] R. Parr and S. Russell. Reinforcement Learning with Hierarchies of Machines. In *Advances in Neural Information Processing Systems 10*, pages 1043–1049. MIT Press, 1997.

[12] R. Sutton and A. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

[13] R. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

[14] M. Grounds, and D. Kudenko. Combining reinforcement learning with symbolic planning. In *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, pages 75–86. Springer, 2008.

[15] S. Russell, and P. Norvig. Artificial Intelligence: A modern approach. In *Artificial Intelligence*. Prentice-Hall, Egnlewood Cliffs, volume 25, pages 27. Citeseer, 1995.

[16] M. Ghallab, D. Nau, and P. Traverso. Automated planning: theory & practice. In Elsevier, 2004.

[17] N. Lipovetzky, and H. Geffner. Width and Serialization of Classical Planning Problems. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI-12)*, pages 540–545, 2012.

[18] R. Jensen, M. Veloso, and R. Bryant. Fault Tolerant Planning: Toward Probabilistic Uncertainty Models in Symbolic Non-Deterministic Planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, pages 335–344, 2004.

[19] N. Koenig, and A. Howard Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-04)*, volume 3, pages 2149–2154, 2004.

[20] L. Kaelbling, and T. Lozano-Pérez Hierarchical task and motion planning in the now. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-11)*, pages 1470–1477, 2011.

[21] J. Wolfe, B. Marthi, and S. Russell. Combined Task and Motion Planning for Mobile Manipulation. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS-10)*, pages 254–258, 2010.

[22] P. Gregory, and A. Lindsay. Domain Model Acquisition in Domains with Action Costs. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS-16)*, pages 149–157, 2016.

[23] D. Martinez, G. Alenyà, C. Torras, T. Ribeiro and K. Inoue. Learning Relational Dynamics of Stochastic Domains for Planning. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS-16)*, pages 235–243, 2016.