Planning Hierarchies and Their Connections to Language

Nakul Gopalan
ngopalan@cs.brown.edu
Brown University

Abstract

Robots working with humans in real environments need to plan in a large state–action space given a natural language command. Such a problem poses multiple challenges with respect to the size of the state–action space to plan over, the different modalities that natural language can provide to specify the goal condition, and the difficulty of learning a model of such an environment to plan over. In this thesis we would look at using hierarchical methods to learn and plan in these large state–action spaces. Further, we would look the using natural language to guide the construction and learning of hierarchies and reward functions.

Introduction

In this work we consider the problem of robots working with humans in real world environments, and try to postulate some solutions that are feasible to solve such problems efficiently. There are many challenges that robot interacting with humans, we specify a few that we try to address in this work. The first challenge is to plan under uncertainty in large state–action spaces, which are continuous. The problem is also exacerbated as the number of manipulable objects in the environment increase, as there is a combinatorial explosion in the state–action space with each object the agent can manipulate. In this thesis we will explore hierarchical methods to solve such tasks.

The second challenge is to follow a natural language command to its goal specification. Natural language allows multiple modalities to present commands. Commands can be specified at different orders of granularity, coarse or fine, allowing a range to specify commands like "get to the library" to "take a left turn". Further, commands can be specified with ends or means of the task as the goal. For example, an instruction to "go to the red room" is very different from "go to the red room through the long corridor." In this thesis we will look at methods that ground natural language commands to reward functions hierarchies or plan directly, depending on the modality demanded by the natural language command.

The third challenge involves learning to solve such tasks efficiently. This involves learning hierarchies and spatio–temporal abstractions that construct the hierarchies. We are interested in looking at connections between attribute learning and option learning to construct these hierarchies. Attribute learning previously has been done using trajectories or natural language. We want to combine these ideas to learn hierarchies, which are efficient to plan over.

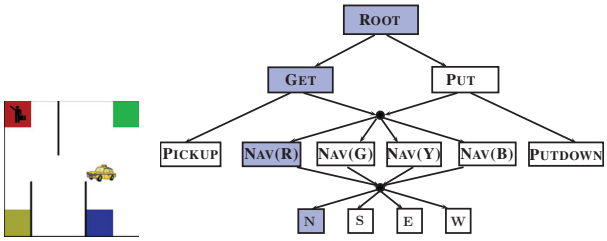
There are other challenges in robotics like partial observability, dialog, vision for robotics, task generalization, etc. which are not the focus of this thesis. In the next sections we would set up the first three challenges in detail along with our proposed solutions.

The Planning problem

When carrying out tasks in unstructured, multifaceted environments such as factory floors or kitchens, the resulting planning problems are extremely challenging due to the large state and action spaces (Bollini et al. 2012; Knepper et al. 2013). Typical planning methods require the agent to explore the state–action space at its lowest level, resulting in a search for long sequences of actions in a combinatorially large state space. For example, cleaning a room requires arranging objects in their respective places. A naive approach for arranging object might have to search over all possible states by placing all objects in all possible locations, resulting in an intractable inference problem with increasing objects.

One promising approach is to decompose planning problems in such domains into a network of independent subgoals. This approach is appealing because the decision-making problem for each subgoal is typically much simpler than the original problem. There are two ways in which the decision problem can be simplified. First, instead of selecting between actions, the agent can select between subgoals that are recursively solved, decreasing the search depth. Second, the state representation of the world can be compressed to include only information that is relevant to the current decision problem. Importantly, planning algorithms for each subproblem can be custom-tailored, allowing each goal to be solved as efficiently as possible.

We proposed *Abstract Markov Decision Process* (AMDP) hierarchies as a method for reasoning about a network of subgoals (Gopalan et al. 2017 in press), we describe the formalism briefly here. AMDPs offer a model-based hierarchical representation that encapsulates knowledge about abstract tasks at each level of the hierarchy, enabling



(a) Taxi (b) Taxi AMDP hierarchy

Figure 1: (a) The Taxi problem, where the taxi needs to drop the passenger to their goal; (b) the Taxi AMDP hierarchy, nodes indicate subgoals which are solved using an AMDP or a primitive action. The edges are actions belonging to the parent AMDP. Shaded nodes indicate which subgoals are expanded by AMDPs in a given state. In contrast, bottom-up approaches like MAXQ (Dietterich 2000) expand all nodes in the figure. These savings result in significant total planning computation gains: AMDP planning requires only 3% of the backups that MAXQ requires for the Taxi problem.

much faster, more flexible top–down planning than previous bottom–up methods like MAXQ (Dietterich 2000) or Options (Sutton, Precup, and Singh 1999). An AMDP is an MDP whose states are abstract representations of the states of an underlying *environment* (the ground MDP). The actions of the AMDP are either primitive actions from the environment MDP or subgoals to be solved. An AMDP hierarchy is an acyclic graph in which each node is a primitive action or an AMDP that solves a subgoal defined by its parent. The main advantage of such a hierarchy is that *only* subgoals that help achieve the main task need to be planned for; crucially, plans for irrelevant subgoals are never computed. Another desirable property of AMDPs is that agents can plan in stochastic environments, since each subgoal's decision problem is represented by an MDP. Moreover, each subgoal can be independently solved by any off-the-shelf MDP planner best suited for solving that subgoal.

For example, consider the Taxi problem (Dietterich 2000) shown in Figure 1a and its AMDP hierarchy in Figure 1b. The objective of the task is to deliver the passenger to their goal location out of four locations on the map. The subgoal of Get Passenger, which picks up the passenger from a source location, is represented by an MDP, with lower-level navigation subgoals, Nav(R), and a passenger-pickup subgoal, Pickup. The state space to solve the Get Passenger subgoal need not include certain aspects of the environment such as the Cartesian coordinates of the taxi and passenger. To solve this small MDP when picking up a passenger at the Red location, it is unnecessary to solve for the subpolicy to navigate to the Blue location. Our hierarchy enables a decision about which subgoal to solve without needing to solve the entire environment MDP.

In this top-down methodology, planning is performed by first computing a policy for the root AMDP for the current projected environment state, and then recursively computing the policy for the subgoals the root policy selects. In contrast a bottom up planner like MAXQ or options based based planning would compute value functions over the hierarchy by processing the state–action space at the lowest level and backing up values to the abstract subtask nodes. This *bottom-up* process requires full expansion of the state–action space, resulting in large amounts of computation.

Moreover, since the tasks are abstractly defined (for example, "take passenger to blue location"), changing the task description from the "blue" to the "red" location is straightforward, and users do not have to directly manipulate the reward functions at each level of the hierarchy. This abstraction is useful in robotics, as human users can simply change the top-level task description and the required behavior will be achieved by the hierarchy.

Formally, we define an AMDP as a six-tuple ($\tilde{\mathcal{S}}$, $\tilde{\mathcal{A}}$, $\tilde{\mathcal{T}}$, $\tilde{\mathcal{R}}$, $\tilde{\mathcal{E}}$, F). These are the usual MDP components, with the addition of $F : \mathcal{S} \rightarrow \tilde{\mathcal{S}}$, a *state projection* function that maps states from the environment MDP into the AMDP state space $\tilde{\mathcal{S}}$. Additionally, the actions ($\tilde{\mathcal{A}}$) of the AMDP are either primitive actions of the environment MDP, or are associated with subgoals to solve in the environment MDP. The transition function of the AMDP ($\tilde{\mathcal{T}}$) must capture the expected changes in the AMDP state space upon completion of these subgoals. With these action and state semantics, an AMDP, in effect, defines a decision problem over subgoals for the environment MDP.

Naturally, each subgoal for a task must be solved. However, even a single subgoal might be challenging to solve in the environment MDP. Therefore, we introduce the concept of an AMDP hierarchy $H = (V, E)$, which is a directed acyclic graph (DAG) with labeled edges. The vertices of the hierarchy V consist of a set of AMDPs $\mathcal{M}$ and the set of the primitive actions $\mathcal{A}$ of the environment MDP. The edges in the hierarchy link multiple AMDPs together, with the edge label associating the action of an AMDP with either a primitive environment action or a subgoal that is formulated as an AMDP itself. Consequently, an AMDP hierarchy recursively breaks down a problem into a series of small subgoals.

We now describe planning with a hierarchy H of AMDPs. The critical property of our planning approach is to make decisions online in a top-down fashion by exploiting the transition and reward function defined for each AMDP. In this top-down methodology, planning is performed by first computing a policy for the root AMDP for the current projected environment state, and then recursively computing the policy for the subgoals the root policy selects. Consequently, the agent never has to determine how to solve subgoals that are not useful subgoals to satisfy, resulting in significant performance gains compared to bottom-up solution methods. This top-down approach does require that the transition model and reward function for each AMDP are available.

If each AMDP's transition dynamics accurately models the subgoal outcomes, then an optimal solution for each AMDP produces a recursively optimal solution for the whole problem; if the transition dynamics are not accurate, then the error associated with the overall solution can still be bounded as shown in our previous work (Gopalan et al. 2017 in press). Further, as each sub-goal has a local model, we can ground any sub-goal in the DAG depending on the

Algorithm 1 Online Hierarchical AMDP Planning

```
function SOLVE(H)
    GROUND(H, ROOT(H))
function GROUND(H, i)
    if i is primitive then                          ▷ recursive base case
        EXECUTE(i)
    else
        s_i ← F_i(s)                                ▷ project the environment state s
        π ← PLAN(s_i, i)
        while s_i ∉ E_i do                          ▷ execute until local termination
            a ← π(s_i)
            j ← LINK(H, i, a)                       ▷ a links to node j
            GROUND(H, j)
            s_i ← F_i(s)
```

task specification as shown in the next section.

Pseudocode for online hierarchical AMDP planning is shown in Algorithm 1. Planning begins by calling the recursive *ground* function from the root of H. If node i passed to the ground function is a primitive action in the environment MDP, then it is executed in the environment. Otherwise, the node is an AMDP that requires solving. Before solving it, the current environment state s is first projected into AMDP i's state space with AMDP i's projection function F_i. Next, any off-the-shelf MDP planning algorithm associated with AMDP i is used to compute a policy. The policy is then followed until a terminal state of the AMDP is reached. Following actions selected by the policy for AMDP i involves finding the node the actions links to in hierarchy H, and then calling the ground function on that node. Note that after the ground function returns, at least one primitive action in the environment should have been executed. Therefore, after ground is called, the current state for the AMDP is updated by projecting the current state of the environment with F_i.

Planning with AMDPs shows significant improvements in planning times when compared with traditional bottom-up planners or flat planners when tested across different domains as shows in the results of (Gopalan et al. 2017 in press). We also showed a real time planning application for task and motion planning in robotics. In this demo a Turtlebot moved a block to from one room to the goal room in presence of environmental disturbances as shown in our video[1]. This is a hard planning problem with a continuous state–action space, and stochasticity in the environment. The agent shows reactive control to retrieve the block in the video as soon as it is snatched, to move the block to the goal room. For more details please refer (Gopalan et al. 2017 in press).

Hence AMDPs show significant improvements in planning times across multiple domains, even with continuous state–action spaces. Now that we have a tool to plan in large domains, we look next at natural language as an input and the different modalities of inputs, some of which would find the use of AMDP hierarchies useful.

[1]https://youtu.be/Bp3VEO66WSg

Goal specification with Natural Language

Natural language provides an easy interface for an untrained public to work with robots. Such robots that understand natural language commands must at the very least understand goal based commands that ask the robot to achieve a certain goal configuration. Abstraction is important for achieving such goal conditions because it is much harder to map natural language to a sequence of robot control signals. Instead existing approaches map natural language commands to a formal representation at some fixed level of abstraction (Chen and Mooney 2011; Matuszek et al. 2012b; Tellex et al. 2011). While effective at directing robots to complete predefined tasks, mapping to fixed sequences of robot actions is unreliable when faced with a changing or stochastic environment. Accordingly, (MacGlashan et al. 2015) decouple the problem and use a statistical language model to map between language and robot goals, expressed as reward functions in a Markov Decision Process (MDP). Then, an arbitrary planner solves the MDP, resolving any environment-specific challenges. As a result, the learned language model can transfer to other robots with different action sets so long as there is consistency in the task representation (*i.e.*, reward functions). However natural language problem specification has different different kinds of requirements: granularity, means and ends of task solving, and temporal specification of goals.

First is the aspect of granularity. For example, a brief transcript from an expert human forklift operator instructing a human trainee has very abstract commands such as "Grab a pallet," mid-level commands such as "Make sure your forks are centered," and very fine-grained commands such as "Tilt back a little bit" all within thirty seconds of dialog. Humans use these varied granularities to specify and reason about a large variety of tasks with a wide range of difficulties. Furthermore, these abstractions in language map to subgoals that are useful when interpreting and executing a task. Moreover, MDPs for complex, real-world environments face an inherent tradeoff between including low-level task representations and increasing the time needed to plan in the presence of both low- and high-level reward functions (Gopalan et al. 2017 in press).

To address this problems, we developed an approach for mapping natural language commands of varying complexities to reward functions at different levels of abstraction within a hierarchical planning framework. This approach enables the system to quickly and accurately interpret both abstract and fine-grained commands. Our system uses a deep neural network language model that learns how to map natural language commands to the appropriate level of an AMDP planning hierarchy. By coupling abstraction level inference with the overall grounding problem, we fully exploit the subsequent AMDP hierarchy to efficiently execute the grounded tasks. To our knowledge, we are the first to contribute a system for grounding language at multiple levels of abstraction, as well as the first to contribute a deep learning system for improved robotic language understanding. The results show faster average planning times at all levels of the hierarchy when compared to a base level planner. A demo

of the system can be seen here[2]. The system can accept low level commands like "go north" and high level commands like "take the block to the red room."

Next we would briefly describe other natural language grounding problems that interest us. First is problem of the means and ends of task solving, where a user might specify how to solve a task. For example the trajectory for "go to red room through the blue room" is very different from the trajectory for "go to the red room." This problem can be solved by a language model that recognizes when the means of solving a task are more important and would then plan for the task with different sets of planners. Second is the problem of temporal specification of rewards, where a command might be "go to the red room and then go to the blue room." Here, we can parse the language with Linear Temporal Logic (LTL) and create a non-Markovian reward function, where the reward functions switch as a subtask is complete. This formulation would be important to solve temporally extended tasks with multiple subgoal specifications given by the human user. Abstraction would be important in these LTL specification as solving these behavioral problems as the lowest level of abstraction might be computationally intractable. Next we look at how we might learn these abstractions.

Learning AMDP Hierarchies

The hierarchies that we looked at until now were hand designed, however an agent has to be capable of creating these hierarchical abstractions on its own in the real world. We postulate that natural language provides some clues about the levels of abstraction that a human agent might care about when working with such robots. We have two goals in this section; firstly we need to learn the local models for AMDP hierarchies; secondly a more important goal is to learn an AMDP hierarchy with language and trajectories.

To solve the first part we can use R-max (Brafman and Tennenholtz 2002) on every local model of an AMDP hierarchy. This approach will learn the level 1 models by collecting samples from the environment, but models at every higher level can be learned exactly by sampling from the models learned at level 1. This method would be sample efficient and would enlighten the trade-offs of having a precise, expensive to learn hierarchical model versus a cheap erroneous hierarchical model.

The second and more important goal is to learn an AMDP hierarchy. Konidaris 2016 uses options or temporally extended actions to learn symbols from initiation and termination sets, to create state abstractions and a higher level in the hierarchy. We believe that an important method to learn symbols can be via natural language. Matuszek et al. 2012a learned attributes of objects present in a state to model language and perception together. Borrowing ideas of attribute learning from existing literature, we can create methods to learn symbols and associated abstract states directly from demonstrations, and plan for them using AMDP hierarchies. A simpler idea to test attribute learning might be to learn object parameterized options, akin to parametrized skills, where we learn object attributes with natural language.

This learning method would satisfy most of the goals with respect to an agent in the real world that learns from natural language and example trajectories; plans in real time given a natural language command at varying degrees of granularity and temporal specification.

[2]https://youtu.be/9bU2oE5RtvU

Conclusion

In this work we look at the problems of understanding natural language groundings, learning efficient hierarchies and planning efficiently to have a robot perform tasks real time in stochastic and large state–action spaces. Our initial results show that the planning problem can be made easier with AMDP hierarchies. We have made some inroads in the natural language grounding problems, where we can specify problems at different levels of granularity to an agent. However, we still have to make large amounts of progress in the problem of learning of a hierarchy. We believe our methods would lead to faster learning of hierarchies and shorter planning times when compared to traditional methods.

Acknowledgements

This material is based upon work supported by the National Science Foundation under grant number IIS-1637614 and the National Aeronautics and Space Administration under grant number NNX16AR61G.

References

Bollini, M.; Tellex, S.; Thompson, T.; Roy, N.; and Rus, D. 2012. Interpreting and executing recipes with a cooking robot. In *International Symposium on Experimental Robotics*.

Brafman, R. I., and Tennenholtz, M. 2002. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3(Oct):213–231.

Chen, D. L., and Mooney, R. J. 2011. Learning to interpret natural language navigation instructions from observations. In *AAAI Conference on Artificial Intelligence*.

Dietterich, T. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Gopalan, N.; desJardins, M.; Littman, M. L.; MacGlashan, J.; Squire, S.; Tellex, S.; Winder, J.; and Wong, L. L. 2017 in press. Planning with abstract markov decision processes. In *International Conference on Automated Planning and Scheduling*.

Knepper, R.; Tellex, S.; Li, A.; Roy, N.; and Rus, D. 2013. Single assembly robot in search of human partner: Versatile grounded language generation. In *ACM/IEEE International Conference on Human-Robot Interaction Workshop on Collaborative Manipulation*.

Konidaris, G. 2016. Constructing abstraction hierarchies using a skill-symbol loop. In *International Joint Conference on Artificial Intelligence*.

MacGlashan, J.; Babeş-Vroman, M.; desJardins, M.; Littman, M. L.; Muresan, S.; Squire, S.; Tellex, S.; Arumugam, D.; and Yang, L. 2015. Grounding English commands to reward functions. In *Robotics: Science and Systems*.

Matuszek, C.; FitzGerald, N.; Zettlemoyer, L.; Bo, L.; and Fox, D. 2012a. A joint model of language and perception for grounded attribute learning. *arXiv preprint arXiv:1206.6423*.

Matuszek, C.; Herbst, E.; Zettlemoyer, L.; and Fox, D. 2012b. Learning to parse natural language commands to a robot control system. In *International Symposium on Experimental Robotics*.

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

Tellex, S.; Kollar, T.; Dickerson, S.; Walter, M. R.; Banerjee, A. G.; Teller, S.; and Roy, N. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI Conference on Artificial Intelligence*.