Learning High-Level Planning from Asking Questions

Adam Yala, Nicola Greco, Sebastien Boyer
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{adamyala,ngreco,sebboyer}@mit.edu

Abstract

Asking questions is an essential tool that humans use to learn. In this paper, we explore the use of questions to model the dynamics of the world in text-aware, high-level planning agents. In contrast to previous work, our planning agent starts with no text and acquires information live by asking questions. Our model jointly learns what question to ask, when to ask them, how to interpret natural language answers, and finally how to perform high-level planning. We train our model via the reinforcement learning framework, using plan success as a reward signal. Our work extends the Minecraft high-level planner proposed by Branavan et al. (2012), and introduces the question answering tool, implemented as a simple word matching between questions and sentences in our knowledge base. We experiment different methods, report our work in progress, showing a 4% improvement on the baseline, where no text is given.[1]

1 Introduction

Asking questions is a natural behavior that we, humans, use to learn about our world. Can this help for artificial intelligent systems? In this paper, we propose to build AI agents more robust to the lack of information. Particularly, we explore the possibility of asking questions about the dynamics of the world, while solving problems. This work can be useful in the context of agents collaborating with humans as well as groups of AI agents exchanging information with each others.

Pre-conditions are the logical requirements to reach states in the world. For example, in the sentence "wood is required to build a wooden door", *wood* is a precondition to *wooden door*. Previous research shows that planning agents can learn these pre-conditions directly from a text given a-priori (Branavan et al., 2012). These techniques performs better than having no text, and nearly as good as knowing the gold pre-conditions. This is particularly useful in the context of complex problems for which text information can be provided a-priori.

Current approaches are limited to the amount of text that is provided in advance. Complex problems often require breadth in understanding the dynamics of the world. Although we could provide all the important text a-priori to make our high-level planner robust, it can be difficult to collect all the potentially useful information.

The central idea of our work aims is to make planning agents robust to lack of information by asking questions. We design a planning agent that can jointly learn what question to ask, when to ask questions, how to extract pre-conditions from the answers obtained and finally to perform high-level planning. In this paper, we extend the high-level planner proposed by Branavan et al. (2012) which consist in training a model via the reinforcement learning framework, using feedback automatically obtained from plan execution attempts. We augment the current actions space with questions. The high-level planner can now sample questions as subgoals, described in Section 3. While action sub-

[1] Our implementation is available on https://github.com/flare-ai/learning_from_questions

goals are executed by the low level planner MetricFF, we send our questions to our Question Answering system, described in Section 4. We sample subgoals and classify precondtion-condition relationships from text using two simple log-linear models. Both are updated as shown in (2) and (3) through plan execution attempts. We show how a very simple model can outperform on the baseline where no text is provide, in Section 6 and 7. Finally, we discuss areas of improvement and further work, in Section 8 and 9.

2 Related Work

The previous work on question can be split in two directions: resolution of ambiguity, dealing with failure.

Resolution of Ambiguity. Tellex et al. (2014) explored using question asking as a way to resolve grounding ambiguity in agents in the context of instruction following robots. A robot might be given the instructions to move a steal plate to the truck. Suppose there are two trucks in the robots environment. The robot must now deal with ambiguity. The authors tackle this problem by asking questions. More specifically, they explore two models for identifying when a question is necessary (entropy on the grounding level, and entropy on the planning level), and ask questions via simple templates, similarly to our approach. The robot then receives a natural language response and is more able to completely tackle the task.

Dealing with failure. It is not uncommon for an agent to be unable complete a task. What a human might do in this case is ask for help. In Tellex et al. (2014), researchers explore how a robot could generate an effective ask for help, to best empower humans to make a difference. More specifically, she applies an inverse mapping from her ground graph for instruction following to a question.

However, compared to previous research, we are tackling a different problem, guiding the planning process itself.

3 Background

Our goal is to build a question asking agent that can use answers to better understand and act in a complex world. In this section, we define the terminology of our problem.

3.1 Predicates, States and Actions

In our system, we combine predicates, $x_i \in X$, to represent the state of our system, $S = \{s \subset X\}$ (e.g. $have(wood)$ $have(stone)$).

Executing action, $a \in A$, from state, s, leads to a new state, s', according to a fixed distribution $T(s'|s, a)$. For each problem, g, the agent is given start state, $s_0^g \in S$, and an end state, $s_f^g \in S$.

3.2 Learning High-Level Planning

The high-level plan is a sequence of single-term predicates (also called *sub-goals*). Transition between sub-goals can have several actions and can are executed by a low level planner. We enforce a 1 minute time limit on low level planning, and any uncompleted plan is counted as a failure. If a transition between two sub-goals is too complex or impossible, the transition fails. For the purpose of this paper the low level planner used is Metric-FF (Hoffmann and Nebel, 2011) and we will consider as a black-box.

Given a set of goals, a planning agent learns to map appropriate high-level plans to goals. Particularly, our agent uses the reinforcement learning framework to learn how to sample a sequence of sub-goals appropriate to a given goal. Using the outcome of a given sequence of sub-goals (did all the transition between sub-goals succeeded ?) for a particular goal g, the agent updates its belief of the distribution $p(\vec{x}|s_0^g, s_f^g)$ where $\vec{x}$ stands for the sequence of sub-goals. We assume a Markovian structure of sub-goal sequences.

3.3 Pre-conditions

As previously mentioned, our agent tries to learn pre-conditions in order to make appropriate plans. The aim is to predict whether sub-goal x_i is a precondition sub-goal x_j. We encapsulate the current knowledge of the agent about all preconditions in a binary square matrix C (where $C[i, j] = 1$ if and only if x_i is a precondition for x_j). Equation 1 describes the sub-goal sequence distribution used by the planning agent, where θ_x parametrizes the distribution (learnt by the agent) and n is the length of the sub-goal sequence.

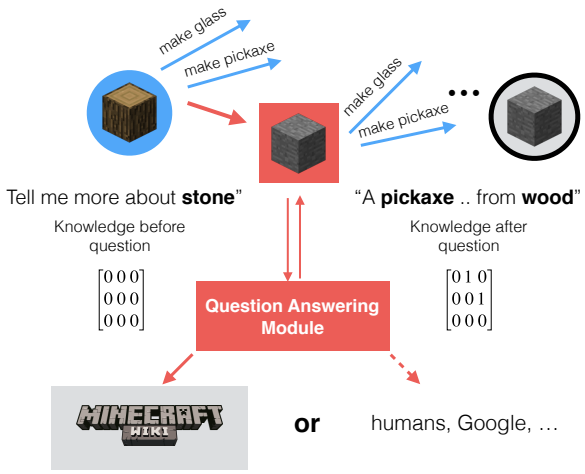

Figure 1: The above is an example of our system. The blue circle containing *wood* is a sub-goal, the square containing *stone* is a question, finally the grey circle is the final goal. In this example, the final goal is "Make a *pickaxe*". At the initial step, our system has no knowledge of pre-conditions (hence the empty pre-conditions matrix, C). At this point in time, the planner could decide to ask about *stone* and after retrieving answer sentences from the Question Answering module, we can update C to finally make better decisions.

$$p(\vec{x}|s_0^g, s_f^g, C; \theta_x) = \prod_{t=1}^{n} p(x_t|x_{t-1}, s_0^g, s_f^g, C; \theta_x)$$

4 Question-asking model

The central challenge of our work is to learn how to acquire the right information at the right time for each problem. In this Section, we present how we designed the question asking model and a simple implementation of an external questioning answering tool. An overview of how our system works is presented in Figure 1.

4.1 Asking questions

The Minecraft vocabulary is restricted to 131 distinct symbols, these are divided into 56 Objects (e.g. *wood*, *pickaxe*, etc.), 68 Actions (e.g. *craft-wood-pickaxe*) and 7 Predicates (e.g. *player-at-map*, *crafting*).

Most of the problem goals are Planning Domain Definition Language (PDDL) formulas composed of one or two terms, for example: "have n amount of object x" (in PDDL syntax $thingAvailable(x) = n$). Given this template of goals, we decided to generate two different templates for our questions:

1. "Tell me about $object$"
2. "Tell me how to $action$"

The use of question based templates was motivated by the recent work (Tellex et al., 2014) in resolving groundings through template based questions. Finally we generate the list of possible questions using template and term pairs and we add these to the vocabulary of the planning agent. In this way, we introduce new question sub-goals that can be sampled either separately or together with the other sub-goals as we discuss in Section 4.2. Normal sub-goals (like *have pickaxe)* are sent to the low level planner and questions are sent to our Question Answering Module described in Section 4.3.

4.2 Two question-asking schemes

To do this, we explored several different approaches.

1. Embedding questions within the sub-goal policy In this approach, we expand the state-space of the sub-goal policy and treat questions as special types of sub-goals. The only difference between questions and sub-goals is that questions must be "solved" by asking the IR system, and sub-goals are sent to a low level planner. Questions and sub-goals share the same feature space, and the policy now has to learn over a 40% larger state space. This comes with a couple caveats. Whereas sub-goals are all sampled, then all solved, questions must be solved immediately, in order to update the knowledge base (the pre-conditions matrix C) and impact the choice of subsequent sub-goals within the same problem.

2. Creating and learning a separate question model given the initial and target states. In this approach, we build a separate log-linear model to pick which questions to ask with it's own state space and feature space. This model conditions over the initial state, and target problem, and generates a short sequence of questions. The agent then learns from

these questions (updating the preconditions matrix, C) before starting to plan.

3. Creating and learning a separate question model over initial, target, and sub-goal sequence. In this approach, we take (2) and also condition over the attempted sampled sub-goal sequence. After answering the questions, we then re-sample the sub-goal sequence.

4.3 Answering Questions

As an end goal, we would like to build collaborative agents that ask question and get answers from humans and other agents. To simulate these helpful humans, we built a simple Question Answering system: questions are answered by a set of k sentences in the Minecraft's Wiki that contain the action or the object of interest. To achieve this we build an Inverted Index for each relevant word and the sentences it appears in.

More specifically, we stemmed and transformed each question object to a natural language name. An example of this would be the simple transformation from iron-door to iron door. We then built indexes to these words to sentences in the corpus. When asked a question, we would select some matching sentences to return at random. This had several favorable properties. First, it keeps our results comparable to Branavan et al. (2012)'s results, since the maximum amount of information that can be acquired is equivalent to the amount given a-priori in the original paper. Secondly, it is cheap to generate answers, allowing our system to leverage hundreds of iterations and learn our model via reinforcement learning.

In summary, our core contribution is a framework to ask and answer questions as part of the planning process.

4.4 General learning framework

We describe our two main design approaches for our question asking system in the pseudo codes that follow. Algorithm 1 describes the first method, where we embedded questions into the sub-goal policy. Algorithm 2 describes the second method, where we learn a different model to ask questions and we ask then at the beginning of each task. We note that the third method will be described by a slightly modified version of Algorithm 2 where we first sample a sequence of sub-goals $\vec{x}$, then ask N questions based on $s_0^g, s_f^g, \vec{x}$, and finally re-sample a sequence $\vec{x}'$ based on the updated C.

Algorithm 1 Question-As-Subgoal method

Input :
A document d, Set of plannings tasks G,
reward function $r(.)$, Number of iterations T
A question answering system $Q(.)$

Initialization :
Model parameters $\theta_x = 0$ and $\theta_c = 0$

Learning :
for i=1...T **do**
  Reinitialize knowledge
  $C \leftarrow 0$
  for $g \in G$ **do**
    Try to complete task g
    for $t = 1...n$ **do**
      Sample new subgoal/question
      $x_t \sim p(x|x_{t-1}, s_0^g, s_f^g, C; \theta_x)$
      if x_t is a question about y **then**
        Ask and receive answer
        $s \leftarrow Q(x_t)$
        for $x \in X$ **do**
          $v_{x,y} \leftarrow p(x \rightarrow y|s, \theta_c)$
          $v_{y,x} \leftarrow p(y \rightarrow x|s, \theta_c)$
          if $v_{x,y} = 1$ **then**
            $C = C \cup \{x, y\}$
          if $v_{y,x} = 1$ **then**
            $C = C \cup \{y, x\}$
      else
        Execute task $x_{t-1} \rightarrow x_t$
    Update parameters in θ_c
  Update parameters in θ_x

Here we describe the equations corresponding to the respective parameters updates. The equations are the same for both methods though equation 4 only matters for method 2 (and ϕ_q should be conditioned on $\vec{x}$ as well when used in method 3.).

$$\Delta\theta_c \leftarrow \alpha_c r[\phi_c(x_i, x_j; w_k, q_k) - \mathbf{E}\left[\phi_c(x_{i'}, x_{j'}; w_k, q_k)\right]] \quad (2)$$

Algorithm 2 Question-First method

Input :
A document d, Set of plannings tasks G,
reward function $r(.)$, Number of iterations T
A question answering system $Q(.)$, the number of questions to ask first N

Initialization :
Model parameters $\theta_x = 0,\ \theta_c = 0$ and $\theta_q = 0$

Learning :
for i=1...T **do**
  Reinitialize knowledge
  $C \leftarrow 0$
  for $g \in G$ **do**
    Ask N questions first
    for $i = 1...N$ **do**
      Ask and receive answer
      $q_i \sim p(q|x_{t-1}, s_0^g, s_f^g, C; \theta_q)$
      $s \leftarrow Q(q_i)$
      for $x \in X$ **do**
        $v_{x,y} \leftarrow p(x \rightarrow y|s, \theta_c)$
        $v_{y,x} \leftarrow p(y \rightarrow x|s, \theta_c)$
        if $v_{x,y} = 1$ **then**
          $C = C \cup \{x, y\}$
        if $v_{y,x} = 1$ **then**
          $C = C \cup \{y, x\}$
    Try to complete task g
    for $t = 1...n$ **do**
      Sample new sub-goal
      $x_t \sim p(x|x_{t-1}, s_0^g, s_f^g, C; \theta_x)$
      Execute task $x_{t-1} \rightarrow x_t$
    Update parameters in θ_c and θ_q
  Update parameters in θ_x

$$\Delta\theta_x \leftarrow \alpha_x r \sum_t [\phi_x(x_t, x_{t-1}, s_0^g, s_f^g, C) - \mathbf{E}\left[\phi_x(x'_t, x_{t-1}, s_0^g, s_f^g, C)\right]] \quad (3)$$

$$\Delta\theta_q \leftarrow \alpha_q r [\phi_q(s_0^g, s_f^g) - \mathbf{E}\left[\phi_q(s_0^g, s_f^g)\right]] \quad (4)$$

5 Technical Challenges

In developing this research, we faced three main challenges. In this Section we will discuss them in order of importance: (1) Code Complexity, (2) Feature Engineering, (3) Learning.

5.1 Code Complexity

Our work is built upon an extensive code base that spans over 70.000 lines of code written in C++. Understanding the different part of code was an intense work of reverse engineering. Most of the time, the language was a barrier to a small tests or experiments, slowing down development cycles. However, we succeeded in introducing the ability of asking questions into the current system, hence setting the ground for further developments.

5.2 Feature Engineering

The existing work relies on large amounts of features engineered to work very well with the current *action* sub-goal system. Extending the feature space by adding questions does not fit well with the existing feature engineering. Depending on how we structure the question PDDL, this causes the system to either ask too few questions or have difficulty learning to differentiate between questions and sub-goals.

5.3 Learning

Given our agent is likely to only have any one specific line of text available for short periods of time during only some iterations, we face several large challenges

Learning to interpret text

It is difficult for the agent to learn how to extract the precondition, condition relationships. The previous system leveraged success and failure of each

sub-goal pair to better tune it's understanding of the text (as described in Equation 3). In our system, sentences, which are sporadically acquired, have less opportunity to have reward attributed to their interpretations.

Learning to weight the acquired-preconditions

It is also difficult to learn to weight specific precondition condition features as important if they are sporadically acquired through questions. Given all weights are initialized to zero, if our system acquires some useful information, our log linear model may fail to use it and fail during the first iteration. The model would then tune its weights, but might not ask that same question again for many iterations.

6 Experiments

To evaluate the effectiveness of our approach, and the fundamental challenges of the problem, we ran several experiments. We discuss each of them bellow.

- *Random questions*. We tried asking 5 random questions at the beginning of each problem. This was done to evaluate the importance of asking the right question for a problem in our domain. We tried this both returning all responses and 5 responses.

- *Simple heuristics* As a next step, we developed an experiment to ask about the end goal at the beginning of each problem. This was done to establish a baseline to see how well simple methods perform. This approach is already developed, but awaiting results due to some technical issues with our server. As an example, if the end goal was *have pick axe*, the heuristics experiment would ask *tell me about the object pick axe*.

- *Approach 1* We use one log linear model and one combined state space to sample questions and sub goals. Each question would get 5 sentences as a response.

- *Approach 2* We use two separate log linear models and two state spaces to sample questions and separately. We condition sampling questions over the initial state, and target goal.

Method	% Plans
FF	40.8
No Text	69.4
Full Model	**80.2**
No Answers	69.4
With Answers	**73.4**

Table 1: The first part of the Table shows Branavan et al. (2012) models: (1) FF, the low-level planner with no text, (2) No Text, the high-level planner with no text, (3) Full Model, the high-level planner with text. The second part shows our results without Answers (comparable to No Text) and with Answers

  This approach is still under development, and waiting for results.

- *Approach 3* We use two separate log linear models but also condition over a sampled sub goal sequence. We condition sampling questions over the initial state, and target goal and one pass of the subgoal model. This approach is still under development, and waiting for results.

- *Decoupling the text interpretation problem* We run each of the above experiments but with a pre-trained text-interpretation model. We do this to evaluate how well our model can learn to ask the right questions.

7 Results

Our primary motivation was to develop more robust agents by empowering to ask questions. In that regard, we have shown results. The agent, without a-priori information, is able to solve 69% of the problems. Our agent with question asking (as described in Approach 1) is able to solve 73%. In proving how robust we can make our planner with question asking, we have only taken a step. We are still actively tackling engineering challenges and developing approaches 2 and 3. Our results are shown in Table 7 compared to Branavan et al. (2012) original results.

8 Future Work

There is a lot of opportunity to explore richer architectures along several dimensions.

- **Learning paradigms**. We can explore richer learning architectures, with deep reinforcement

learning (Narasimhan et al., 2015) or any technique to avoid manual feature engineering.

- **Information sources**. Our work can be extended by evaluating the relevance of different sources of information for a particular world (as well as mixture of those sources). For the *Minecraft's* world, one could use human answers from online forums or *Google* search engine in addition tot the *wiki* we used.

- **World complexities**. Testing our ideas in more complex world would inform how scalable our emthod can be for real life tasks. A simple way to do it would be to increase the size of the *Minecraft's* world given to the agent. But we also think about testing our scheme on more complex games such as *Civilization*.

9 Conclusion

We explored the use of questions to help high-level planning agents model the dynamics of the world. We reverse the paradigm of using a-priori text-information to guide agents, to an online paradigm, where agents acquire information live as they need it. Our agents jointly learn what questions to ask, when to ask them, how to interpret natural language answers and how to perform high-planning. We have shown that planners can be made more robust to lack of information, and surpass agents with no a-priori information, if we extend them with basic question asking. When bench-marked on a small Minecraft world, question asking allows our agent to complete 4% more goals. We consider our work to be a first step toward a new paradigm for text-aware planners. We still have several experiments in the pipeline and aspire to influence the way for new research endeavors in this area.

Acknowledgments

We would like to thank S.R.K Branavan, Professor Regina Barzilay, Karthik Narasimhan and Master Patch for their help and suggestions.

References

[Branavan et al.2012] SRK Branavan, Nate Kushman, Tao Lei, and Regina Barzilay. 2012. Learning high-level planning from text. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 126–135. Association for Computational Linguistics.

[Hoffmann and Nebel2011] Jörg Hoffmann and Bernhard Nebel. 2011. The FF planning system: Fast plan generation through heuristic search. *CoRR*, abs/1106.0675.

[Narasimhan et al.2015] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015. Language understanding for text-based games using deep reinforcement learning. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1–11. The Association for Computational Linguistics.

[Tellex et al.2014] Stefanie Tellex, Ross A. Knepper, Adrian Li, Daniela Rus, and Nicholas Roy. 2014. Asking for help using inverse semantics. In Dieter Fox, Lydia E. Kavraki, and Hanna Kurniawati, editors, *Robotics: Science and Systems X, University of California, Berkeley, USA, July 12-16, 2014*.