Adaptive Agents in Minecraft: A Hybrid Paradigm for Combining Domain Knowledge with Reinforcement Learning

Priyam Parashar
Contextual Robotics Institute
Jacobs School of Engineering
University of California, San Diego
La Jolla, CA 92093

Bradley Sheneman
American Family Insurance
Chicago, IL

Ashok Goel
School of Interactive Computing
Georgia Institute of Technology
Atlanta, Georgia 30332–0250

***Abstract*—We present a pilot study focused on creating flexible Hierarchical Task Networks which can leverage Reinforcement Learning to repair and adapt incomplete plans in the simulated rich domain of Minecraft. This paper presents an early evaluation of our algorithm using simulation for adaptive agents planning in a dynamic world. Our algorithm uses an hierarchical planner and can theoretically be used for any type of "bot". The main aim of our study is to create flexible knowledge-based planners for robots, which can leverage exploration and guide learning more efficiently by imparting structure using domain knowledge. Results from simulations indicate that a combined approach using both HTN and RL is more flexible than HTN alone and more efficient than RL alone.**

***Keywords*—reinforcement learning, artificial intelligence, simulation, adaptive agents**

I. Introduction

Hierarchical Task Networks (HTNs) have been used extensively in artificial intelligence applications, specially robotics. They have many advantages which make them a lucrative choice for programming task-level behaviors in structured environments. The biggest one, of course, is that by being hierarchical they are invariant to the low-level agent controllers, which enables re-use of successful plans and simplifies programming. Furthermore, they impose a symbolic or object-oriented structure on world, making higher-level task planning and reasoning easier. Interestingly though this is also one of the major flaws of HTN planner. Classic HTNs follow a strictly structured approach towards task planning, making them incapable of handling dynamic environments. In this paper we experimentally show a way of making HTN plans more flexible by introducing reinforcement learning in the system which can learn new plan segments by exploration. The benefit of such an approach is two-fold, reinforcement learning helps networks learn new plans in dynamic environments and domain knowledge helps guide reinforcement learning towards more fruitful states for faster convergence with fewer data samples.

The obvious question here is why is making HTNs flexible an important endeavour? The short answer is because everyday life is full of chaos and noise. The long answer is that embodied intelligent agents are rapidly moving from industrial sector to personal ones. Robots are being employed in offices, universities, hospitals, etc. to name a few places. These environments are highly dynamic and require agents to be more adaptable and flexible with their assumptions. Agents which can learn, either by demonstrations or exploration, are therefore heavily explored and favored for automating workflow in named domains [1, 5, 6, 18].

Fig. 1. A view of the Puzzle Room showing the Agent and Gold separated by a Glass Wall

The reason for machine learning gaining ground is the appeal for customization of the robot. Traditionally robots need a very structured environment for reliable accuracy in work which results in substantial setup time, and in some cases a procedure overhaul, before introducing a robot to the workflow. This process is very different from how human workers are expected to function. When moving from one to another similar environment, humans tend to explore the surroundings by actions or asking questions until they build a better mental model. Machine learning helps robots by learning from experimentation or demonstrations by non-experts which reduces manual coding effort of experts. Reinforcement learning has been seen as one of the most successful unsupervised methods of learning optimal goal-directed behavior in an unknown environment. The biggest critique of this method has been that reinforcement learners need a substantial amount of domain knowledge, or huge amounts of data, to efficiently understand, manipulate and examine the world and results of actions. This leads to an obvious marriage between the two methods outlined

above.

In this paper, we have built an artificially intelligent agent capable of higher-level reasoning and borrowing knowledge from known problems to solve new ones by employing a guided reinforcement learner. We borrow our intuition from the key concept of *scaffolding* in cognitive science. Scaffolding, in its oldest definition[4], means to highlight the actions of master or the learner which contribute more to the success of a task. In our experiments, our agent is asked to plan course of actions for achieving a certain goal in some scenario. The agent has some prior knowledge of solving a similar problem in a different situation. Our algorithm basically compares the new situation to the most similar known problem, and uses the differences along with domain knowledge from its knowledge base to guide exploration of the reinforcement learner by providing rewards or discounts for fruitful actions. As of right now we are providing the most similar known problem manually to the system, leaving the rest of the reasoning up-to the algorithm.

We are exploring two key concepts here. The main hypothesis is that we can use the domain-knowledge stored in HTN to help guide RL better and speed up its learning curve. The other hypothesis concerns focusing of attention at the right level of detail. HTNs by definition are hierarchical and we hypothesize that this information can help in further focusing attention on the right actions to better explore the environment. We elaborate this point in more detail in the approach section. We would like to point out here that implementing this algorithm on real embodied agents would have required substantial effort in implementing accurate perception, manipulation, etc. While our focus in this paper is to verify our approach and methods first, before adding other unstable components to the pipeline. We have therefore used simulation in this paper for verifying our concept and evaluating the algorithm.

II. Literature Review

Hierarchical Task Networks have been extensively explored in the AI research community in the last few decades, owing to its expressivity[29], speed and efficiency in complex domains, and invariance to lower-level mechanics of execution [9, 20, 21]. Specifically, HTNs have been popular in robotics due to its ability to re-use plans[30] and accurate task planning in structured domains [15]. Given the complexity of real-world scenarios, the symbolic abstractions used by HTNs can measurably speed up the planning time[30]. Apart from manipulators, HTNs have been successfully used in improving navigation strategies for mobile robots by reasoning on future actions of the robot[3]. HTNs are also discussed in human-robot interaction community, specially human-guided learning. Humans tend to think of tasks in a naturally hierarchical way, and HTNs have been seen as a fitting format to learn these representations[17].

Reinforcement Learning (RL) [25] is a well known machine learning technique for training appropriate agent behavior using the concept of rewards. The technique is influenced by concepts from psychology where subjects, especially young children, are rewarded for appropriate behavior and penalized for inappropriate actions to help them learn the norms of culture and society [2, 23]. In machine learning, this technique is used to provide appropriate reward to the agent depending upon consequences of its actions. This helps the agent learn the correct actions to be taken in different conditions or *states*, as an indirect way of learning the correct cost function associated with the environment and the task. Recently, learning game-playing policies using only visual cues has gained much traction in the community due to its obvious benefits in an unstructured domain[16].

Reinforcement Learning has also seen an increased interest from the robotics community in the last decade. Especially it has been observed that model-based versions of RL seem to do exceptionally well in robotics[13]. Trying to merge together new knowledge with known knowledge-base is not a new endeavour and has been extensively explored in literature. Cognitive scientists recognize that rules coded using higher-level knowledge can help guide lower-level actions for better skill acquisition[24]. In the field of AI, Murdock and Goel [19] used model-based reasoning to localize and guide RL, while Ulam et al [28] propose fusing RL with domain-knowledge in video games to improve training efficiency. Other authors have modified a flavor of HTN to calculate and update beliefs of success for different methods, and improve re-planning by focusing on the more successful plans[10, 14]. Hogg and Nejati propose algorithms to create HTNs in a way such that non-determinism is baked-into the methods by first observing task demonstrations[11, 22]. Minecraft platform itself is a very new phenomenon in aiding and exploring different learning methods in the community and [26] is an important recent paper relevant to our mission, highlighting the versatility and ease of use of the platform.

III. Approach

A. Hierarchical Task Networks

Hierarchical Task Networks (HTNs) [8, 9] are one of the more classic approaches used in the world of planning, especially robotics. HTNs represent the environment in terms of a dictionary of symbolic state variables and plans. This includes a library of *primitive actions* and *methods*. A primitive action is the smallest unit of plan decomposition. A method is a composite action made up of one or more ordered primitive actions or methods. It comprises of two main attributes: *pre-conditions* and *effects*. Pre-conditions are a set of environment conditions conditioned on state variables which must be true for a method to be executed. Effects are changes that the method, if executed, would have on the environment variables. Depending upon the goal and the state of the environment HTNs string together these methods to build a complete plan.

In the current context, the HTN uses atomic actions like ”move forward”, ”turn left/right” and ”break block in focus”. For all experiments in this paper the end-goal of the agent remains same, which is to acquire the gold block.

B. Reinforcement Learning: Q-learning

We have implemented a tabular form of Q-learning for our reinforcement learning purposes in this paper, using the following update formula. s denotes a state from the table, a denotes the action taken in state s, s' symbolizes the next

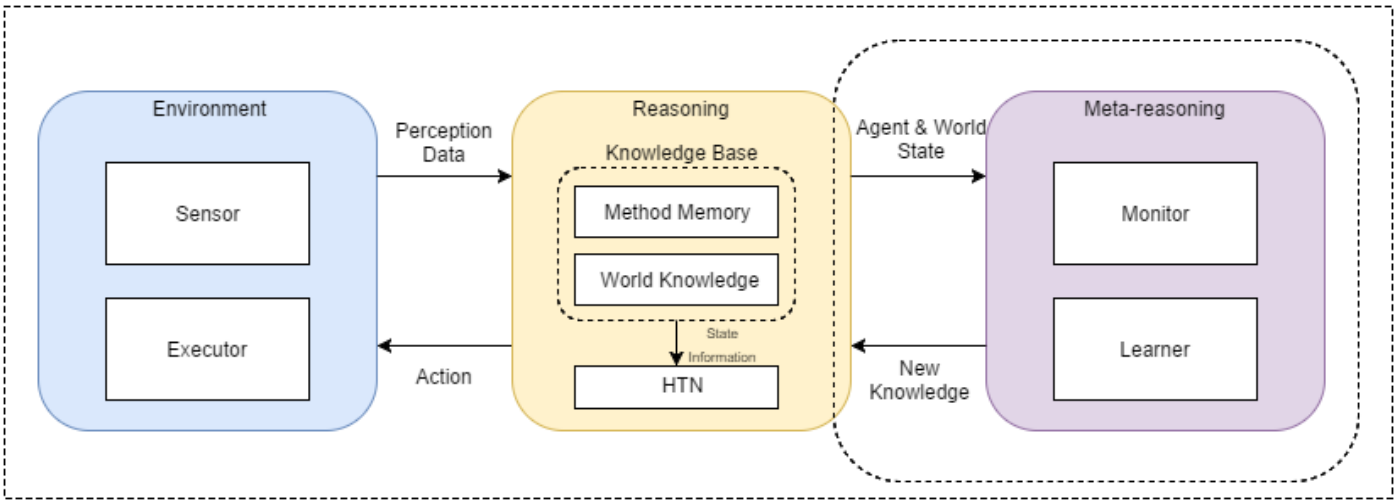

Fig. 2. System Architecture

state once action a is executed and $R(s, a)$ is the reward agent received after executing action a while in state s.

$$Q(s,a) = Q(s,a) + \alpha * (R(s,a) + \gamma * \arg\max_{a'} Q(s',a') - Q(s,a)) \quad (1)$$

The states of the table vary depending upon whether the q-learner is using domain-knowledge or not. World of Minecraft is grid-based and the pure q-learner states consist of the 9 blocks right in front of the agent including the ground blocks, what the agent is staring at, what is the agent holding in its hand and agent's pitch state, i.e. angle at which the agent is staring. For the combination learner, we have also provided states with a count of relevant items within a 5x5 grid around the agent. We have used ϵ-greedy selection strategy, with an exponentially decaying ϵ. After some calibration, our implementation uses a starter ϵ of 0.4 with a decay rate of 0.95 over 1000 iterations, a learning rate, α, of 0.55 and a γ of 0.75. In addition, the Q-values were normalized so as to sum to 50. The ϵ decays as per the following formula, where `decay_steps` is 100 in our implementation:

$$new_\epsilon = starter_\epsilon * \texttt{decay\_rate}^{\frac{\texttt{iteration\_step}}{\texttt{decay\_steps}}}$$

C. Architecture

The architecture, Figure 2, is divided into three major components: Environment Interface, Reasoning AI and a Meta-Reasoner. This three-layered architecture is similar to traditional AI architectures for metareasoning [7, 12]. The environment interface consists of the actual game engine and API, where agent takes actions and uses sensors to perceive surroundings. The simulation is achieved using the rich world of Minecraft with the help of Malmo Platform[1]. The reasoning AI is the actual planner (HTN in our case) that communicates directly with the environment and reasons on environment state and agent-specific variables to build and execute a plan. We have used a stripped down python version of SHOP [21] called PyHop[2] in our implementation. The meta-reasoner is the third hidden component which communicates with the AI and keeps track of internal processes responsible for planning and execution with the help of internal meta-data like error flags and execution trace of planning process. This part emulates the process of debugging run-time error using meta-information as well as deploying a solution just like human developers. The solution, in our implementation, is the Learner module which uses the information provided by meta-reasoner to setup rewards for appropriate states for Q-learning.

D. Algorithm

As noted above in sub-section III-A, the agent is continually processing current world state with method pre-conditions (within the Reasoning component) before en-queueing any action execution. In a dynamic world, this is where the first break happens. The reality is different from the expected. This raises an error flag followed by compilation of an error message, including level of mismatch, rest of the plan and name of mismatched method. This information is dispatched to the Meta-reasoner, which uses it to grab the pre-conditions of methods queued after the mismatched method in the plan. These states are used as intermediate states or goals for the learner, intuition being that if the learner can find a way to these states the planner can re-use the coded methods to achieve the goal. Moreover, the Meta-reasoner forms a

Fig. 3. A view of the compared rooms

[1]https://github.com/Microsoft/malmo
[2]https://bitbucket.org/dananau/pyhop

comparison of current scenario to the nearest known scenario encountered in the past which it knows the solution to (Figure 3).

This comparison helps in creating a secondary level of rewards which is endowed on those actions which make this scenario more like the one already known and solved. A third layer of discounts is formed by looking at or being near *relevant items*. These relevant items are defined by the differences between current and compared scenario and the knowledge base. Any action which leads the agent in direct line of sight of relevant items or brings the agent near relevant items is discounted by some amount. We are discounting the cost, i.e. such a fruitful action costs -0.5 as compared to -1 of normal action, and not rewarding it because we still want the agent to maximize overall reward with minimum number of actions. An example of fruitful action can be seen in Figure 4. Once the q-values are converged above a threshold or once the agent achieves the goal more than a threshold number of times, this learned policy is then added to the method library with the mismatched set of state variables as its pre-conditions.

Let us further clarify with the help of an example. Figure 9 is an example of a plan proposed by the HTN for a scenario where the agent needs to break the wall before acquiring the gold block. During run-time though, the agent realizes that the scenario is modified and the plan breaks down while processing `break_the_wall` method. This raises an error flag and a stack-trace is generated describing the breakage point and the reason for breakage. Using these messages, the meta-reasoner deduces that the pre-conditions for `break_the_wall` method were not satisfied. It then looks ahead and grabs the pre-conditions of methods queued after the named method. Meta-reasoner then uses these grabbed pre-conditions to generate reward states for the agent and deploys the RL module which explores the simulated world to learn a new method to bridge the broken plan.

IV. Experimental Setup

Our experimental setup borrows from the classic ”room solving” puzzle games which required the player to solve a level by acquiring gold or reaching the exit door by overcoming certain obstacles. For our experiment we created three similar puzzles with varying levels of complexity. We have kept things relatively simple in our puzzle rooms in order to verify our concept rather than robustness of the system. As in the classic puzzles, in this experiment the room is considered solved when the agent successfully acquires the gold block.

Fig. 4. An example of a fruitful action where the agent is directly staring at the Glass wall

Looking at the decomposition of `break_the_wall` method more carefully in figure 9, we can see that not only does this method have pre-conditions specific to execution conditions (for example, do not trigger until agent is right next to the wall) but it also has some *inventory* pre-conditions which require the presence of certain tools for successful action execution. There could be two scenarios here, either that the agent was not able to successfully navigate to the wall, say because of a ditch, or the agent did not have the required tools to successfully execute the method. These two problems require two completely different solutions. While the first scenario might require learning of a whole new method to traverse a ditch, the other only requires playing with different tools to find a valid substitute. This is where the hierarchical nature of HTNs helps guide the learner towards right nature of solution. Depending upon whether the breakage was due to new environmental conditions or agent's limited experience with different artifacts, the Meta-reasoner deploys different kinds of solutions to repair the knowledge-base of the planner.

We thus created two different classes of experimental scenarios to test the hierarchical nature of learning from our system. One class tests the adaptability of methods, by rendering an inventory-listed tool unavailable to the user forcing the agent to improvise by learning a new tool on the fly. The other class operates on problems one level above, changing the world state such that none of the stored methods match the current state, rendering a stored method invalid for our scenario. The agent is then instructed to explore the world and learn an alternate method to achieve its immediate goal.

A. Adapting a Known Method

Using our wall-in-the-room setup, we placed the agent and the gold block on opposite sides of this stone wall. We first wrote a plan in which agent uses an “iron axe” tool to first break the wall to access the gold block. To introduce the agent to a new situation, we changed agent's inventory to have a “wood axe” and “steel axe” instead. Thus everything else remains the same except that the agent now has different tools than the one planned for.

B. Learning a New Method

We created an environment that was new for the agent but similar to a known scenario stored in HTN memory. We used a simple empty-pair-of-rooms plan to solve a Puzzle Room with a wall in the moddle, as can be seen in Figure 1. We want to test if the agent can learn the full method from scratch.

C. Combination Learner Versus End-to-End Learner

Finally, we want to compare the efficiency of such an architecture versus one which can not reason about the failure of a plan and decides to employ an end-to-end learner which learns a complete plan from breakage point to the final goal. For this we create a new pipeline and run it on the same wall-in-the-room scenario. Instead of reasoning about information gap and learning a bridging method, this pipeline follows a brute learning policy by employing a learner which learns a

completely new method from point of failure with its goal as gold block acquisition. We then compare the training time and resultant accuracy between this brute end-to-end learner pipeline with our results from our architecture.

Require: s_a: State of the Agent, s_w: State of the World, $method_t$: Current method to be executed, htn_plan
1: **procedure** PLANNER()
2: **while** $method_t \neq \varnothing$ **do**
3: let PC = EXTRACTPRECONDITION($method_t$)
4: **if** s_w = PC **then**
5: EXECUTE($method_t$)
6: UPDATESTATE(s_w)
7: $method_t \leftarrow$ NEXTMETHOD(HTNPlan, $t+1$)
8: **else**
9: new_method $\leftarrow$ METAAI(s_w, s_a, htn_plan, $method_t$)
10: ADDNEWMETHOD(htn_plan, new_method)
11: **end if**
12: **end while**
13: **end procedure**

Fig. 5. Central Planning and Execution Algorithm

Require: s_a, s_w, htn_plan, $method_t$
1: **procedure** METAAI()
2: error_level $\leftarrow$ FINDERRORLEVEL(htn_plan.error_msgs)
3: **if** error_level = PreconditionMismatch **then**
4: actions $\leftarrow$ EXTRACTALLACTIONS(htn_plan.library)
5: intermediate_states $\leftarrow$ EXTRACTPRECONDITIONS(All methods in htn_plan queued after $method_t$)
6: $R(s) \leftarrow$ SETUPREWARDS(intermediate_states)
7: INITIALIZE(QLearner, s_w, $R(s)$, actions)
8: QLEARNER.ADDSTATEVARIABLE(relevant_item_count)
9: LAUNCH(QLearner)
10: **else if** error_level = InventoryMismatch **then**
11: relevant_actions $\leftarrow$ EXTRACTINVENTORYACTIONS(htn_plan.library)
12: intermediate_state $\leftarrow$ EFFECTOFMETHOD($method_t$)
13: $R(s) \leftarrow$ SETUPREWARDS(intermediate_state)
14: INITIALIZE(QLearner, s_w, $R(s)$, relevant_actions)
15: LAUNCH(QLearner)
16: **end if**
17: **end procedure**

Fig. 6. Meta-reasoner Algorithm

V. OBSERVATIONS, RESULTS AND DISCUSSION

Figure 7 shows the comparison between end-to-end learner and our combination learner for the two different method learning scenarios. We would like to point out an interesting observation here, when we compared HTN enriched RL agent with pure RL agent, the pure RL agent resulted in zero percentage of success in completing the mission over 1000 iterations. Our theory is that the proposed scenario was a little too complicated for a simple algorithm like zero-order

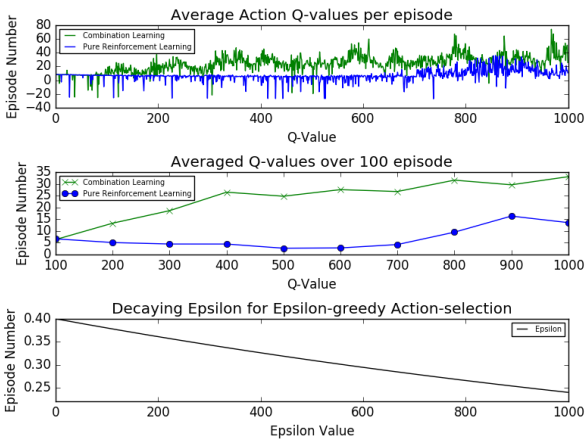

Fig. 7. Comparing action Q-values for different approaches

tabular Q-learning to formulate. The solution required three different actions strung in a row together without missing a beat, which was hard for a no-memory technique to make tractable. Therefore, the results that we show are contrasting between pure RL enriched with room comparison rewards and HTN enriched RL with room comparison as well as fruitful action discounts.

As readers can see in Figure 7, the Q-values for pure reinforcement learning approach first take a dip before gaining value. This is due to the agent's repeatedly wrong or unfruitful actions which further decrease its confidence in actions. The topmost plot shows a considerable amount of spikes and jumping around for the Q-values, this is because the ϵ for our action-selection strategy is still pretty high with a lowest value of 0.25. This leads to execution of random actions by the agent, but since our environment's solution relying on a strictly sequential series of actions even one wrong random action can lead the agent down a rabbit-hole with no gains. The most important results can be seen in the second subplot in the figure, where our combination learner performs significantly better than the pure reinforcement learner. We have used an averaged plot of Q-values here to account for randomness introduced by moderately high ϵ value and to display the comparison more clearly.

Our results are very much in line with the findings of Ulam et al[27, 28] where they saw a considerable speed-up of learning process by providing it with internal model and knowledge about the game world. However, our algorithm goes a step beyond the reactive nature of learning described in the paper and outlines an automated way of mining out relevant reward information from successes of the past to promote a deliberative flavor of learning. The proposed approach is also simpler to implement as compared to [11] which requires complete bottom-up construction of new plans. With memory becoming cheap and processing power available in the cloud, our approach holds merit with its quick learning curve. As can be seen in Figure 8, our agent learns to stay alive for longer quicker, in terms of number of iterations, than pure RL agent. This is important and interesting. Such an observation indicates that even if the agent is not yet proficient in solving the puzzle, it has learnt the boundaries of absolute failure. This is helpful when the sustenance of agent and prolonged exploration is key

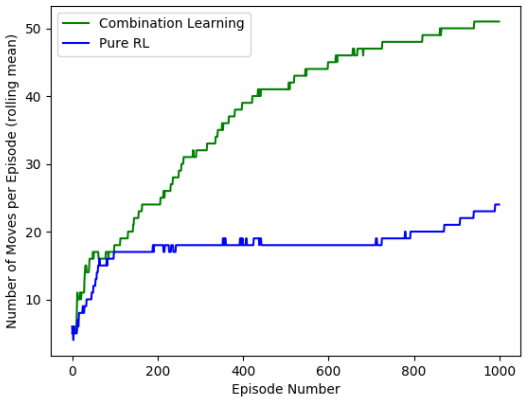

Fig. 8. Number of Moves taken by agent per episode

to learning better solutions.

Moreover, the combination of HTN and RL ends up being more flexible than either HTN or pure RL methods. While classical HTNs are by nature inflexible, and reinforcement learning being very specific to its start and end state, this modular approach lets us re-use the small chunks of methods in any arbitrary sequence to form a plan. The observations of Tessler et al [26] agree with this claim. The authors have used another hierarchical planner with an advanced flavor of reinforcement learning in their paper which helps support claims about generality of this combination of techniques.

We have not shown the result for first class of experiments since it merely involved simulating same action with different inventory items. Since we were selecting item in a randomised manner, the efficiency, in terms of speed of finding the correct item, was not a measurable evaluation criteria.

VI. Conclusion

As shown in this paper, using a dedicated diagnostic system and a meta-reasoning component can measurably increase the efficiency of planning systems. This is important because this enables us to have a flexible planner capable of extending and repairing its knowledge base. Such an ability makes it easier for the industry and consumers to use adaptive agents which come with pre-built domain information, ready to work out-of-the-box as well as capable of tweaking that information as per the changes specific to new environment. They can plan well for the situations already seen, and can potentially learn for new situations by exploring. This is a critical missing piece towards enabling agents to handle open-world situations. Additionally, talking from a computation perspective, the guidance that the HTN provides to machine learner helps scope the to-be-explored state-space by a big factor. Such an architecture beats end-to-end learners not only in terms of efficiency but also flexibility, since methods can be strung together in any order to accomplish different tasks.

VII. Future Work

Our first item of action is to evaluate this algorithm on an actual embodied agent working in real-world scenarios. We also plan on using richer planning languages with our architecture, which not only are stronger at planning but also provide better diagnostic information of the working of a system. As we said earlier, diagnostics are the backbone of our meta-reasoning component. Rich diagnostic messages, apart from providing internal information, can also be leveraged to create templated explanations to the users to elaborate the purpose of an action. Machine learning techniques, generally, tend to absorb patterns from data in the form of mathematical policies and functions and usually can not explain the purpose or reason for learnt behavior. By such a hybrid approach, we plan to use the internal diagnostic information along with meta-reasoning layer to be build an interactive learner which can not only exploit exploration but also knowledge from human users to adapt to new situations.

References

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, DOI: 10.1016/j.robot.2008.10.024.

[2] W. C. Becker, *Parents are teachers: a child management program.* 1971.

[3] T. Belker, M. Hammel, and J. Hertzberg, "Learning to optimize mobile robot navigation based on htn plans," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, IEEE, volume 3, 2003, pages 4136–4141.

[4] L. E. Berk and A. Winsler, *Scaffolding Children's Learning: Vygotsky and Early Childhood Education. NAEYC Research into Practice Series. Volume 7.* ERIC, 1995.

[5] C. Breazeal, D. Buchsbaum, J. Gray, D. Gatenby, and B. Blumberg, "Learning from and about others: towards using imitation to bootstrap the social understanding of others by robots," *Artificial Life*, 2004.

[6] C. Breazeal and B. Scassellati, "Robots that imitate humans," *Trends in cognitive sciences*, volume 6, number 11, pages 481–487, 2002.

[7] M. T. Cox and A. Raja, *Metareasoning: Thinking about thinking*. MIT Press, 2011.

[8] K. Erol, J. A. Hendler, and D. S. Nau, "Umcp: A sound and complete procedure for hierarchical task-network planning.," in *AIPS*, volume 94, 1994, pages 249–254.

[9] K. Erol, J. Hendler, and D. S. Nau, "Htn planning: Complexity and expressivity," in *AAAI*, volume 94, 1994, pages 1123–1128.

[10] H. Hayashi, S. Tokura, T. Hasegawa, and F. Ozaki, "Dynagent: an incremental forward-chaining htn planning agent in dynamic domains," 2006.

[11] C. Hogg, U. Kuter, and H. Muñoz-Avila, "Learning hierarchical task networks for nondeterministic planning domains," *IJCAI International Joint Conference on Artificial Intelligence*, pages 1708–1714, 2009.

[12] J. K. Jones and A. K. Goel, "Perceptually grounded self-diagnosis and self-repair of domain knowledge," *Knowledge-Based Systems*, volume 27, pages 281–301, 2012.

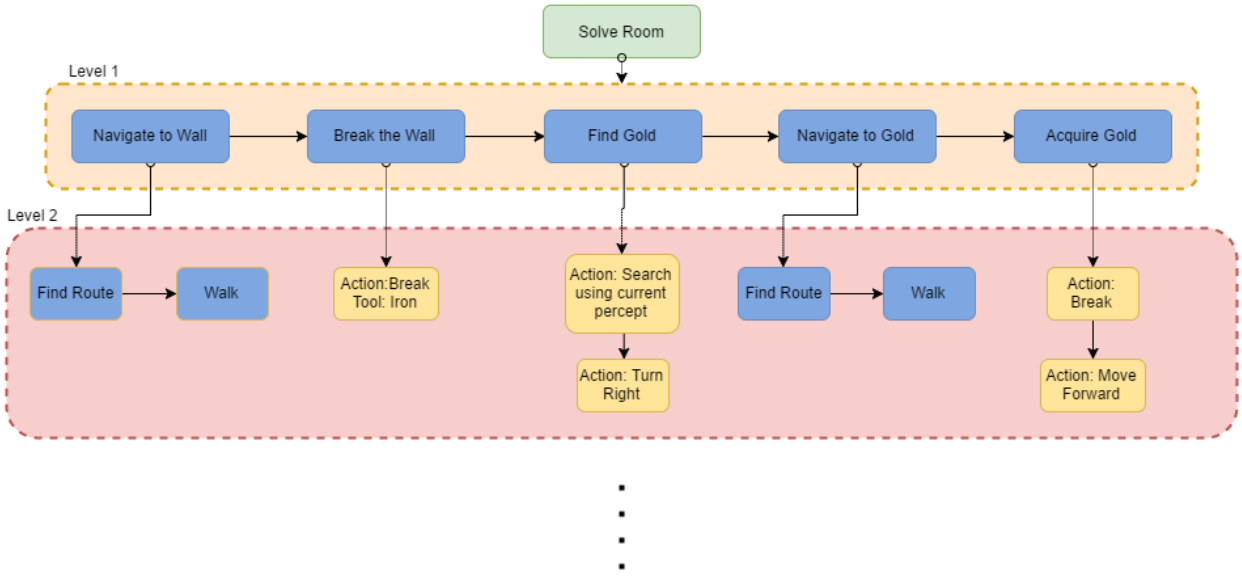

Fig. 9. Plan made for the Wall-in-the-Room Scenario by HTN Planner. Blue boxes symbolize methods and yellow boxes symbolize primitive or atomic actions. This diagram only shows one level of expansion of the plan for explanation purposes, the blue boxes on Level 2 can still be expanded further.

[13] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: a survey," *International Journal of Robotics Research*, volume 32, number 11, pages 1238–1278, 2013. [Online]. Available: http://repository.cmu.edu/robotics.

[14] S. Magnenat, J.-C. Chappelier, and F. Mondada, "Integration of online learning into htn planning for robotic tasks.," in *AAAI Spring Symposium: Designing Intelligent Robots*, 2012.

[15] S. Magnenat, M. Voelkle, and F. Mondada, "Planner9, a htn planner distributed on groups of miniature mobile robots," in *International Conference on Intelligent Robotics and Applications*, Springer, 2009, pages 1013–1022.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *ArXiv preprint arXiv:1312.5602*, 2013.

[17] A. Mohseni-Kabir, C. Rich, S. Chernova, C. L. Sidner, and D. Miller, "Interactive hierarchical task learning from a single demonstration," in *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, ACM, 2015, pages 205–212.

[18] J. Morimoto and K. Doya, "Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning," *Robotics and Autonomous Systems*, volume 36, pages 37–51, 2001.

[19] J. W. Murdock and A. K. Goel, "Meta-case-based reasoning: Self-improvement through self-understanding," *Journal of Experimental & Theoretical Artificial Intelligence*, volume 20, number 1, pages 1–36, 2008.

[20] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, "Shop2: an htn planning system," *Journal of Artificial Intelligence Research*, volume 20, pages 379–404, 2003.

[21] D. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila, *Shop: simple hierarchical ordered planner*, 1999.

[22] N. Nejati, P. Langley, and T. Konik, "Learning hierarchical task networks by observation," in *Proceedings of the 23rd international conference on Machine learning*, ACM, 2006, pages 665–672.

[23] J. B. Sidowski, L. B. Wyckoff, and L. Tabory, "The influence of reinforcement and punishment in a minimal social situation.," *The Journal of Abnormal and Social Psychology*, volume 52, number 1, page 115, 1956.

[24] R. Sun and X. Zhang, "Top-down versus bottom-up learning in cognitive skill acquisition," *Cognitive Systems Research*, volume 5, number 1, pages 63–89, 2004.

[25] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 1. MIT press Cambridge, 1998, volume 1.

[26] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in minecraft," *CoRR*, volume abs/1604.07255, 2016. [Online]. Available: http://arxiv.org/abs/1604.07255.

[27] P. Ulam, A. Goel, J. Jones, and W. Murdock, "Using model-based reflection to guide reinforcement learning," *Reasoning, Representation, and Learning in Computer Games*, page 107, 2005.

[28] P. Ulam, J. Jones, and A. K. Goel, "Combining model-based meta-reasoning and reinforcement learning for adapting game-playing agents," *Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 132–137, 2008.

[29] M. Williamson, K. Decker, and K. Sycara, "Unified information and control flow in hierarchical task networks," in *Proceedings of the AAAI-96 workshop on Theories of Planning, Action, and Control*, 1996.

[30] J. Wolfe, B. Marthi, and S. Russell, "Combined task and motion planning for mobile manipulation," *International Conference on Automated Planning and Scheduling*, pages 254–257, 2010.