Published in the Knowledge Engineering Review Journal vol 33

Towards Life-Long Adaptive Agents: Using Meta-reasoning for Combining Knowledge-based Planning with Situated Learning

Priyam Parashar

Contextual Robotics Institute, UC San Diego, La Jolla, CA 92093, USA
E-mail: pparashar@ucsd.edu

Ashok K. Goel

Design & Intelligence Laboratory, Georgia Institute of Technology, Atlanta, GA 30308, USA
E-mail: goel@cc.gatech.edu

Bradley Sheneman

American Family Insurance, Chicago, IL
E-mail: bradsheneman@gmail.com

Henrik I. Christensen

Contextual Robotics Institute, UC San Diego, La Jolla, CA 92093, USA
E-mail: hichristensen@ucsd.edu

Abstract

We consider task planning for long-living AI agents situated in dynamic environments. Specifically, we address the problem of incomplete knowledge of the world due to the addition of new objects with unknown action models. We propose a multilayered agent architecture that uses meta-reasoning to control hierarchical task planning and situated learning, monitor expectations generated by a plan against world observations, forms goals and rewards for the situated reinforcement learner, and learns the missing planning knowledge relevant to the new objects. We use occupancy grids as a low-level representation for the high-level expectations to capture changes in the physical world due to the additional objects, and provide a similarity method for detecting discrepancies between the expectations and the observations at run-time; the meta-reasoner uses these discrepancies to formulate goals and rewards for the learner, and the learned policies are added to the HTN plan library for future re-use. We describe our experiments in the Minecraft and Gazebo microworlds to demonstrate the efficacy of the architecture and the technique for learning. We test our approach against an ablated RL version, and our results indicate this form of expectation enhances the learning curve for RL while being more generic than propositional representations.

1 Introduction

Action selection is a central problem for all intelligent agents: given a set of percepts indicating the state of the world, what actions may the agent select to fulfill its goals? Planning is a powerful technique for action selection, wherein an agent uses an abstract model of the world to plan actions on the world to meet a goal (Blum & Furst 1997, Nilsson 1998, 2014). To make the problem tractable, many practical planners make simplifying assumptions about the observability of the world and the determinability of the outcomes of actions on the world. These planners work well for short-living agents in highly engineered closed-world environments where the planner can assume complete and correct knowledge of the world (Nilsson 1998, 2014, Thrun et al. 2005). However, these simplifying assumptions tend to break down in (even partially) unstructured, dynamic and open-ended domains. Given that long-living agents in real worlds almost surely will witness changes in their environments, the question becomes how such agents can adapt their knowledge and actions to the changes?

For unstructured, open-ended environments, AI has developed alternative techniques that typically engage plan adaptation or plan learning. In case-based planning, for example, the agent achieves new goals by adapting previously formed plans to achieve similar goals (Cox et al. 2005, Hammond 2012, Kolodner 2014, Leake 1996, Ontanón et al. 2010, Riesbeck & Schank 2013). However, case-based planning is typically effective only when the new goals are variations of previously encountered goals. In learning by experimentation, the agent explores the world and uses reinforcement learning to learn a policy for taking actions in the world (Sutton & Barto 1998). Unfortunately, reinforcement learning can be computationally costly. In learning by demonstration, the agent learns a plan to accomplish a new goal by observing a human teacher (Argall et al. 2009, Breazeal 2004, Breazeal & Scassellati 2002). However, the problem of transfer of the learned model to new situations in an open-ended world remains unsolved.

Recently, there have been several proposals in AI research for integrating some of the above methods, for example, knowledge-based planning and reinforcement learning. Integrating these methods, the argument goes, may help in combining their advantages, and in using one method to compensate for the limitations of another method. For example, knowledge-based planning may localize and guide reinforcement learning, thereby reducing the computational cost of the latter (Efthymiadis & Kudenko 2013, Grounds & Kudenko 2008, Grzes & Kudenko 2008); on the other hand, reinforcement learning might search or analyze the local world and help fill a knowledge gap in the planner (Efthymiadis et al. 2016, Efthymiadis & Kudenko 2015). However, this raises another question: how may we combine the above methods in a principled manner? Existing techniques use the concept of reward shaping (Ng et al. 1999), meta-reasoning (Murdock & Goel 2008), or goal-driven autonomy (Muñoz-Avila et al. 2010).

In this paper, we describe a multilayered agent architecture that uses meta-reasoning for combining knowledge-based planning and situated learning to learn and adapt solutions to novel situations. In particular, we use Hierarchical Task Networks (HTNs) with modified task representations as for the planning technique and reinforcement learning (RL) as the situated learning method. The planner in our architecture retrieves the most relevant HTN plan from memory and executes it, while the meta-reasoner monitors the execution of the HTN plan, localizes the gap when the plan fails upon execution, invokes the RL method to fill the gap in the HTN knowledge, and thus repairs the HTN plan. While the reason of plan failure can be many, our paper focuses explicitly on changes caused by the addition of new objects with unknown action models to the world.

This work directly builds on our previous work on the REM cognitive architecture for combining meta-reasoning, knowledge-based planning and situated learning (Murdock & Goel 2001). The three significant differences are that (1) while REM used Task-Method-Knowledge models arising from work on knowledge systems, our present work uses HTNs that are more common in AI research on planning; (2) while our previous work addressed problems in which the goal of the new task was a variant of the goal of a familiar task, in our present work the new and the known tasks, which we dub original task, have the same goal but different environmental configurations that break planning assumptions; (3) our current paper introduces a more generic and low-level definition for representing and storing plan expectations for discrepancy calculation using occupancy grids.

However, the present work is limited to the addition of objects in the world that render the agent's world knowledge (correct but) incomplete. Given this condition, the meta-reasoner compares the execution trace of HTN plan for the current task against trace from original task and builds a report describing how do the environmental changes break the object-level assumptions involved. In the current implementation, this means a cell-by-cell comparison of the occupancy grids to note similarities. This report includes the HTN method whose pre-conditions are not met in the current environment, a dissimilarity metric for the extent of mismatch and an array of object-level differences between the two environments. This information is used by the meta-reasoner to create goals and reward heuristics for the reinforcement learner which interacts with the environment of the variant task to learn a new method. Once a new method is learned the meta-reasoner uses the meta-information available to it to plug back the new method with the correct identifiers, or pre-conditions, to fill the gap in the plan from original task. As a result, the learner helps in expanding the method knowledge base of the planner itself.

The remaining paper is organized into an overview of relevant literature followed by sections which go into detail about the methods used and elaborate on our system architecture. We demonstrate this framework for adaptive agents in the micro-world of a Minecraft simulation engine as well as grasp planning for a simulated robot in Gazebo. Our preliminary experiments indicate that the hybrid technique is both faster than pure RL and more flexible than pure HTN in dynamic environments with object additions.

2 Motivating Example

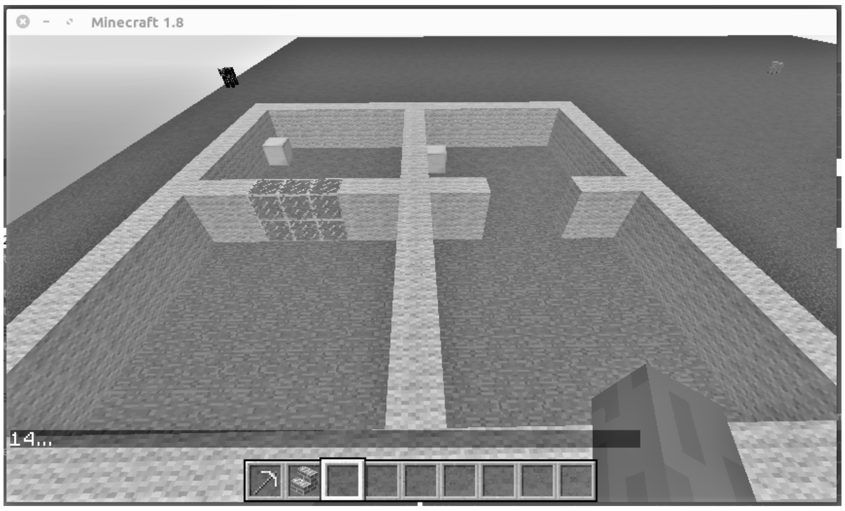

Figure 1 Variant (l) and original (r) Tasks in Minecraft

In order to understand some of the issues and concepts involved, lets consider an agent called Moon in the world of Minecraft. Moon is a rational agent equipped with a planner and knowledge of the primitive actions allowed in the world. Moon is also a mobile agent, i.e., it can maneuver the world and also has a laser-scanner that allows it to “see” the world via laser-scans. A laser-scanner does linear scans of the world at discretized heights and gives back Moon the depth at which it hit an object, and also the reflected value from the object which tell about its material. Our agent lives in a 2-room arena with an open door and has the goal of acquiring a gold block placed somewhere in room 2, i.e., the room further back in Figure 1 above. Moon also has limited knowledge of the world which consists mainly of the objects it expects to see in the arena and their action models, i.e., how do they react to various objects through primitive actions. Moon’s planner also follows a blind planning strategy, i.e., it creates a plan assuming each action is going to be successful. When it finds a plan, it shuts down until invoked again for a new plan.

Moon follows a simple plan that makes it explore the arena until the gold block is sighted, at which point it calls a computationally expensive planner to get a path leading to the gold block. Since Moon has limited resources, it stores the successful plans to re-use. Moon does not want to stray away from the plan, so it also records the state transitions encountered to make sure plan is re-used as expected.

However, Minecraft is a changing world. Imagine there lives a builder who adds new constructions to the world without telling the resident about every small change. The builder decides to build a glass wall in place of the open door without telling Moon or its planner.

This situation is analogous to how we, humans, act in our daily world. We are imprecise and move things around without really announcing every little change. The difference is that for Moon it takes some advanced reasoning to figure out the same change that we humans consider apparent or straightforward to resolve. So now unaware of these changes, Moon starts its task of finding the gold block. However, because of the unexpected glass wall the walking actions of Moon, which lead to room 2, are rendered non-executable. Let's discuss some of the issues here.

Moon does not have any knowledge of the material glass or how to interact with it. The first question is, how to describe this unknown entity to the planner to plan for? Let's say somehow Moon decides this an entirely new scenario and it should learn a resolution for this. What should be the learning goal here? Is it possible for Moon to reuse some of the knowledge and prior plan it already has to formulate better goals and make this process more time-efficient? Finally, let's say Moon does end up learning a policy which leads it to the gold block. How should this policy be represented within its planning framework to re-use in the future? These are a few questions which our algorithm answers in the Approach section.

3 Relevant Work

The topic of meta-reasoning has growing literature in AI research. (Cox 2005) and (Anderson & Oates 2007) provide useful reviews of some of the literature. (Cox & Raja 2011) present an anthology of several research projects on meta-reasoning. Our work builds on the Autognostic, REM, Augur, and GAIA projects.

The Autognostic project (Stroulia & Goel 1995, 1999) developed a multilayered agent architecture including situated, deliberative and meta-reasoning components. In analogy to the redesign of physical devices, the Autognostic system viewed an intelligent agent as an abstract device and used a functional model of the agents design to retrospectively repair its deliberative navigational planner. For example, given a failure of the navigation plan, Autognostic used the functional model of the planner to localize the causes of its failure and invoked an adaptation plan from a library.

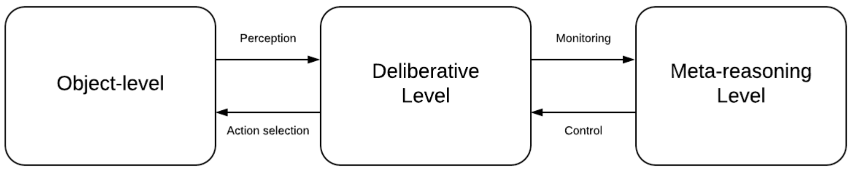

Figure 2 Traditional meta-reasoning architecture

The REM project (Murdock & Goel 2011) built on and generalized the Autognostic agent architecture. It developed a knowledge representation language called Task-Method-Knowledge Language for capturing functional models of agent designs. TMKL is more expressive than HTN; in particular, TMKL sets more explicit expectations about the world states upon completion of tasks, which is useful in localizing the causes of a plan failure. REM addresses both retrospective and proactive adaptations; given a new task such as disassembly of a physical device in proactive adaptation, REM analogically transfers and adapts the reasoning strategy expressed in TMKL for a similar known task such as assembly of the same device. We call this meta-case-based reasoning. Finally, in the retrospective adaption, REM first uses the TMKL model of the assembly planner to localize the causes of a plan failure and then invokes situated reinforcement learning to learn the repair to the planner.

Unlike the Autognostic and REM projects, the Augur project (Goel & Jones 2011, Jones & Goel 2012) focuses on the use of meta-reasoning to diagnose and repair domain knowledge grounded in perception. Given domain knowledge in the form of a classification hierarchy, Augur associates meta-knowledge in the form of empirically verification procedures that capture the expectations about the world states with each node in the hierarchy. When the classifier makes an incorrect prediction, Augur systematically invokes the empirical verification procedures to diagnose the classification knowledge.

Finally, the GAIA project (Goel & Rugaber 2017) provides an interactive CAD-like environment for constructing game-playing agents (for playing Freeciv) in the REM architecture and the TMKL2 language. Given the design of a game-playing agent, GAIA executes the agent in the game environment. If the agent fails, then GAIA enables interactive diagnosis and repair of the agents design.

This present work builds on the above research, and especially the REM architecture and its combination of situated, deliberative, and meta-reasoning strategies. However, unlike previous REM research that focused on variations of tasks assigned to an intelligent while keeping the environment constant, this work keeps the task constant but varies the environment through the addition of objects. It complements a similar project on one-shot learning by demonstration in which a robot is given a single demonstration of a task and must transfer what it learned to the same task in a new environment consisting of additional objects (Fitzgerald et al. 2016). In the other project, the robot asks informed questions of the teacher to identify what knowledge to transfer and how to adapt it to the new environment. In the present work, we take the approach of combining planning with situated learning.

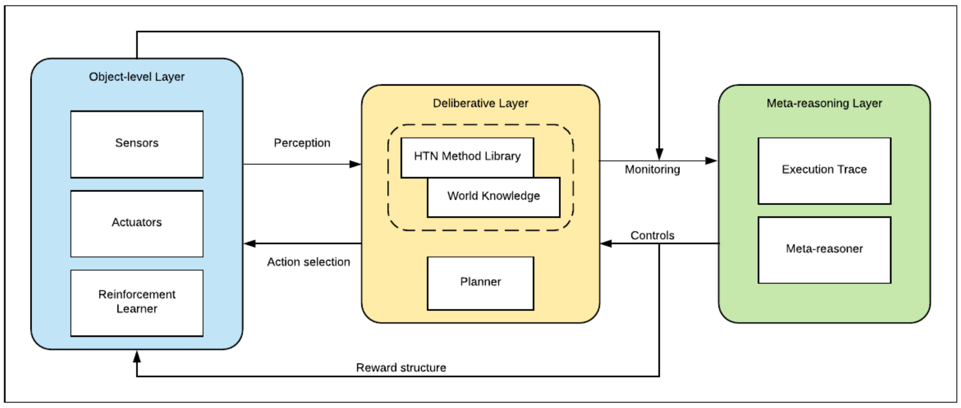

Figure 3 Proposed system architecture: HTN is the planner, RL the learner

GDA (Muñoz-Avila et al. 2010) is a goal-reasoning framework where the agent makes use of an expectation knowledge-base (KB) to keep track of plan execution and find discrepancies if any. GDA is based on meta-reasoning concepts like expectation matching system, discrepancy detection, explanation generation, etc. The central concept of GDA is that goals should have an option to be updated, or reformulated, once plans go awry, based on some reasoning. While (Muñoz-Avila et al. 2010) follow a rule-based goal formulation scheme, LGDA (Jaidee et al. 2011*a*,*b*) learns *(discrepancy, goal)* pairs by experience. (Jaidee et al. 2012) Is perhaps the most advanced GDA formulation where the agent also learns new alternative goals and goal-based policies from experience. GDA generally employs RL as its learning algorithm of choice, which means learning too many alternative goals runs (Jaidee et al. 2012) into the risk of requiring many episodes before becoming competent.

Apart from learning goal formulation, GDA research has also looked at other parts of the pipeline, some of them directly relevant to our current work. (Dannenhauer & Muñoz-Avila 2015*a*) use hierarchical plans with annotated expectations, called h-plans, along with a semantic web ontology, to make better assertions about game states and reason at various levels of the plan hierarchy leading to better control over strategy reformulation. (Dannenhauer & Muñoz-Avila 2015*b*) look at the problem of creating expectations for a plan and introduces informed expectations which are more general than effects of primitive action but less restrictive than full-state comparisons. (Dannenhauer et al. 2016) extend the former to partially observable environments by also considering sensing costs, which is something that our approach also touches on using a low-level variant of expectations, as described in the Algorithm section. (Vattam et al. 2013) provide a survey of research on goal reasoning including goal-driven autonomy.

Another concept proposed by the AI community relevant to our work is of using reward shaping (Ng et al. 1999) to provide a denser reward to the agent which helps in trimming out unnecessary exploration of irrelevant state-space. Most of the approaches using reward-shaping with a knowledge-based plan (Efthymiadis & Kudenko 2013, Grounds & Kudenko 2008, Grzes & Kudenko 2008) to support it come under the umbrella of Knowledge-Based Reinforcement Learning. This relationship also works the other way around where the novel transitions seen during exploration can be used to tag faulty knowledge for revision (Efthymiadis & Kudenko 2015) or to model a new abstract MDP which reflects the unknown knowledge about the environment (Efthymiadis et al. 2016). However, in our paper reward-shaping is not the focus but one of the tools used to incorporate our contribution, i.e., a low-level expectation definition, into a reinforcement learning framework.

4 Background

We are using a modified flavor of Hierarchical Task Networks for our planning and plan representation needs, coupled with a hand-coded retrieval system for plan re-use. Simple, tabular Q-learning is being used for learning from experience when the plan breaks down.

4.1 Modified Hierarchical Task Networks

Hierarchical Task Networks (HTNs) (Erol et al. 1994, Nau et al. 2003, 1999) are one of the more classic, and more widely used, approaches used in the world of planning. HTNs represent the environment regarding a dictionary of symbolic state variables and task 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 two primary attributes: pre-conditions and effects. Preconditions are a set of environmental conditions over state variables which must be true for a method to be executed. Effects are the environmental changes triggered by the methods execution. Depending on the goal and the state of the environment HTNs string together these methods to build a complete plan. HTNs are generally non-flexible and not-ideal for dynamic task environments.

To expand the usage of HTNs to meta-reasoning, we have modified the plan representation to also include meta-data from its last successful run, which will be treated as the planners expectation during run-time. This was done so that apart from using case-based retrieval of a plan, the old execution of the plan can also be used to generate insight into the change in environment between variant task and the original task. We describe the nature of this meta-data or expectation in more detail in the Algorithm section below.

4.2 Reinforcement Learning: Q-learning

We have implemented a tabular form of learning action Q-values for our reinforcement learning purposes in this paper(Sutton & Barto 1998, Watkins & Dayan 1992). We have used a Q-learning update for our experiments using the following formula. The variable s denotes a state from the table, a denotes the action taken in state s , s' symbolizes the next state once action a is executed, a' is the action chosen based on Q-values and action-selection policy of the agent while in state s' and $R(s, a)$ is the reward agent received after executing action a while in state s . Q-learning update equation is formulated as follows:

$$Q(s, a) = Q(s, a) + \alpha \cdot \left(R(s, a) + \gamma \cdot \arg\max_{a'} Q(s', a') - Q(s, a) \right) \tag{1}$$

The rewards in this paper follow a per-action scheme, where each action is given a -1 reward except the terminal state. We chose this scheme so that the agent uses a minimum number of actions to execute the desired behavior. The states of the table vary depending upon whether the q-learner is using domain-knowledge or not, see Section 5 for input state details for both. We have used an ϵ-greedy selection strategy, with a decaying ϵ . The decay is controlled by the following formula, where $decay\_steps$ is 100 in our implementation:

$$\epsilon \leftarrow \epsilon \cdot decay\_rate^{\frac{iteration\_step}{decay\_steps}}$$

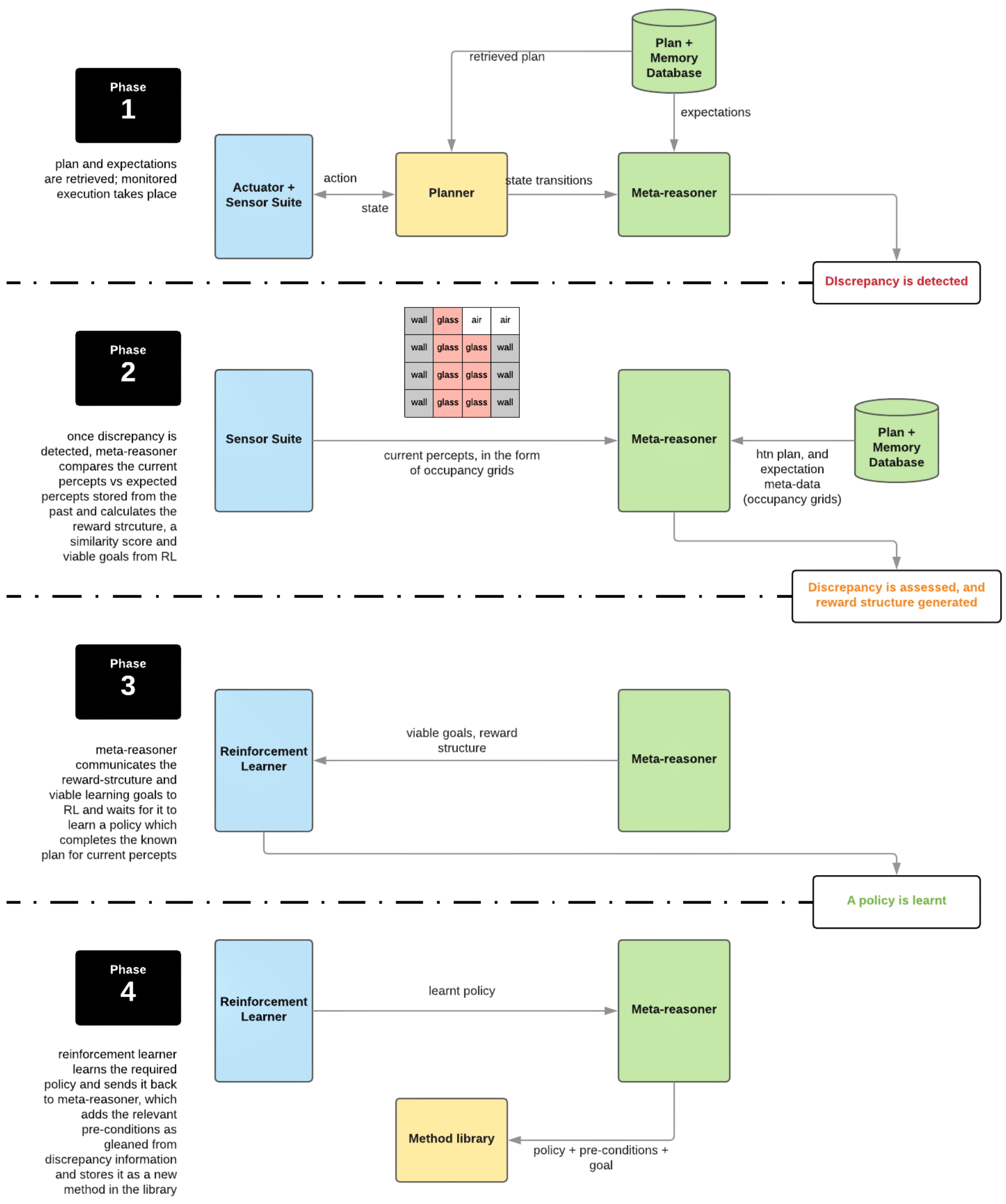
Phase 1
plan and expectations are retrieved; monitored execution takes place
Plan + Memory Database
retrieved plan
expectations
Actuator + Sensor Suite
action
state
Planner
state transitions
Meta-reasoner
Discrepancy is detected
Phase 2
once discrepancy is detected, meta-reasoner compares the current percepts vs expected percepts stored from the past and calculates the reward strcuture, a similarity score and viable goals from RL
wall glass air air
wall glass glass wall
wall glass glass wall
wall glass glass wall
Sensor Suite
current percepts, in the form of occupancy grids
Meta-reasoner
htn plan, and expectation meta-data (occupancy grids)
Plan + Memory Database
Discrepancy is assessed, and reward structure generated
Phase 3
meta-reasoner communicates the reward-strcuture and viable learning goals to RL and waits for it to learn a policy which completes the known plan for current percepts
Reinforcement Learner
viable goals, reward structure
Meta-reasoner
A policy is learnt
Phase 4
reinforcement learner learns the required policy and sends it back to meta-reasoner, which adds the relevant pre-conditions as gleaned from discrepancy information and stores it as a new method in the library
Reinforcement Learner
learnt policy
Meta-reasoner
Method library
policy + pre-conditions + goal

Figure 4 Process Diagram: An outline of the communication between different processes

5 Approach

5.1 Meta-reasoning: Architecture and Concepts

Our proposed architecture follows a note-assess-guide strategy (Cox et al. 2016, Paisner et al. 2013). In a few words, this strategy looks for any changes from expected plan execution and makes a note of the event, next it assesses the nature of the discrepancy, and finally uses relevant knowledge to guide a resolution.

In our architecture (Figure 3), object-level layer consists of the sensor-reading, actuator controls (provided by simulation engine and the APIs) and a reactive Q-learner. The deliberative layer houses the planner, its method library and trace memory of plan execution. This trace involves the whole plan annotated with an ordered history of state transitions for every method. These trace memories are the expectations on which our framework bases most of its decisions. The astute reader will note here that we make a strong assumption on the complete and noise-less observability of the environment. Our

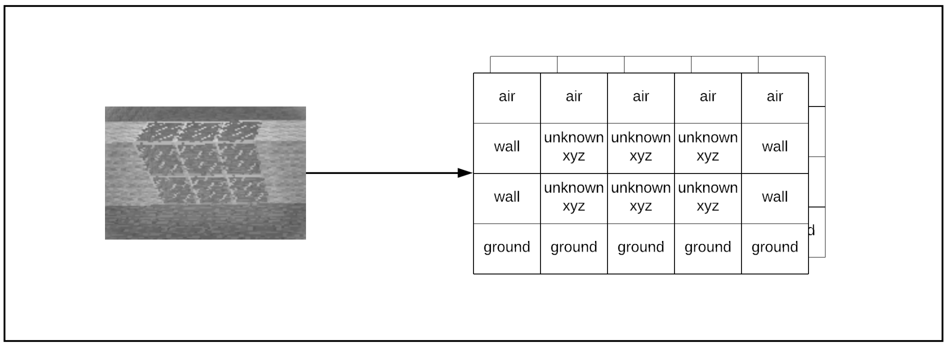

Figure 5 Depiction of state-to-occupancy-grid encoding

expectations are the laser-scans that the sensors emit, and we are using 3D occupancy grids (Figure 5) or representing these sensory expectations. The reason for this representation choice is two-fold: (a) occupancy grids are the data structures of choice in robotics community to compress and represent unstructured data, making our approach more readily suitable to such agents; and (b) since we need to describe unknown entities, a representation which accounts for the physical properties made more sense than using any higher-level descriptors. Both the layers described in this paragraph can communicate with each other.

Our final meta-reasoning layer houses the meta-reasoner and has communication channels to and from both the object-level and deliberative layer. This layer is responsible for monitoring to ensure execution matches expectations. If not, we call the event a discrepancy. Concerning the occupancy grid form of expectations, our discrepancy is defined by a dissimilarity score and a list of cell indices where the grid readings do not match.

5.2 *Algorithm*

Given the birds eye view of our algorithm, lets deep dive into how the components instantiate the note, assess and guide processes. The planning process or phase one starts when the goal is passed to the deliberative layer. If the planner has a stored plan from an original task environment relevant to the current variant task environment, then it is retrieved for execution along with the related expectations sent to meta-reasoner for monitoring. All our agents are equipped with laser scanners which scan a 270-degree area in front of the agent at different heights. Each laser scan tells the agent about the distance of obstacles from itself, and the returned intensity of the laser helps in identifying the material of the obstacle. The state data, originating from these lasers, is communicated to the meta-reasoner at each time-step in the form of occupancy grids. Each cell in the grid has two fields associated with it: a) status: FREE, OCCUPIED or UNKNOWN, and b) reflected reading which tells about the material the laser struck. The height up till which the scans should be considered is specified by the configuration space (Lozano-Perez 1983) of the agent, which is a robotics planning concept used to calculate the volume of interest for a plan in which the obstacles are to be avoided. Usually for planning configuration space is created by padding obstacles to find a safe plan, in our case, however, since we already have the path we pad some space around it to represent the physical body of the robot and only look for discrepancies in that volume.

The meta-reasoner compares the execution sensor readings with expectations to catch any discrepancies. Discrepancies are found by calculating a similarity score between the expectation and the sensor reading for each method. If similarity score is lower than a threshold, a discrepancy report is generated which includes the dissimilarity score, a list of grid cell indices which differ and the pre-conditions of all methods in the partial HTN plan after the current method. Our current work uses an extreme threshold of $simmax$ where if the grids are different by even one grid cell, it gets noted. Considering the availability of low-level snapshots of the environment and keeping in mind that the nature of discrepancy is one of the additions of new objects, it is easy to come up with a comparison metric for calculating the similarity between two different instantiations of the same environment. For the current implementation, we use simple addition, $sim(x, s')$, of a function $g(cube_i^x, cube_i^{s'})$ over all the cubes of both snapshots. The functions are defined as:

$$g\left(cube_i^x, cube_i^{s'}\right) = \begin{cases} 1, if x(cube_i) = s(cube_i) \\ 0, if x(cube_i) \neq s'(cube_i) \end{cases} \tag{2}$$

$$sim(x, s') = \sum_i g\left(cube_i^x, cube_i^{s'}\right) \tag{3}$$

$sim(x, s')$ is lower-bounded at 0 and upper-bounded by the product of resolution of the quantization and physical volume of the snapshot area, or $simmax$. A lower value of this sum indicates a more substantial difference between the two snapshots.

Next is the assess process or phase two, where the meta-reasoner combines the knowledge and experience with reinforcement learning to learn a discrepancy-resolving policy. For a reinforcement learner to be efficient, it requires achievable goals and guiding reward potential. Our algorithm uses the partial plan after the current method where discrepancy occurred as viable goals to seek since the planner already has methods to achieve the overall goals from those states. Additionally, the meta-reasoner also adds a reward strategy where the agent gets a random cost discount (between 0 and tau¡=0.5) for actions which make the environment more like the original task. The intuition for this random reward strategy is that if the agent can manage to make the current environment more like the original, then it should just be able to re-use the plan. Such rewards also encourage interactions with the new entities in the environment indirectly encouraging experimental interactions. These changes are identified by a decrease in the dissimilarity score of the environment. Phase three is the guide process where the reinforcement learner takes the viable goals and rewards and interacts with the environment to learn a discrepancy-resolution policy.

Once the reinforcement learner has learnt a new policy adhering to these goals and rewards (based on some confidence threshold for Q-values or maximum number of interactions), it communicates back to the meta-reasoner alerting it to the availability of a policy for encountered discrepancy and also communicates which of the given viable goals are achieved by this policy. This is the last phase, or phase four of the algorithm, where the meta-reasoner takes the learned policy attaches it to the state where the discrepancy was found as a pre-condition as well as its goal state as an outcome and stores this tuple as a method in HTNs method library for future use.

6 Experimental Setup

We use two different simulation engines with a mobile agent and a robotic arm in Minecraft and Gazebo, respectively. The HTN planner (python version of SHOP(Nau et al. 1999) called PyHop) assumes a separate controller is used for low-level joint movements and only plans for more abstract task primitives. Both the agents use the same meta-reasoning framework but have different task specifications and original-variant environment settings. We have kept the problems relatively simple in our experiments since we wanted to focus on verifying our concept first rather than evaluating the robustness of the system. We present a quick summary of the experiments here to make the details clear to the reader.

6.1 Gold-hunting in Minecraft

This experiment follows directly from our motivating example. The agent lives in a two-room arena and is tasked with finding a gold block in the second room (Figure 11). For original-variant modification, the arenas open door is replaced with a glass wall. We have used the Malmo project for integrating Minecraft environment with rewards for reinforcement learning. The primitive actions in this experiment are: “move forward”, “turn left/right”, “look up/down by 45 degrees” and “break the block in focus”. The input state for pure reinforcement learner consisted of the 3(height)x3(width) 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. the angle at which the agent is staring. The terminating conditions of this experiment were either acquiring the gold and a reward or breaking the walls or ground of the room in which case the agent was considered to have failed. 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 to sum to 50.

6.2 Grasping a can of Coke

Our second experiment is using the Gazebo simulator (Koenig & Howard 2004) along with a robotic arm whose goal is to pick up a can of coke placed on the table-top. We have used Gazebo's API for the reward and environment management, and a Q-learner for reinforcement learning purposes. The primitive actions here are: "pick a specified object" and "place object in hand on the table at (x, y)".

The HTN planner plans a path for picking up the can of coke in an original environment which is free of litter, while in the variant task the configuration space of the path is obstructed by solid cubes. The plan for this experiment is more low-level as compared to the gold-hunting scenario. It consists of an ordered list of 3-dimensional points the arm's gripper should follow before it closes to grasp the can. Therefore, the intermediate goals here are clearing out individual segments of this plan, of the obstacles, for the gripper to move through. Since the arm is an immobile and fixed agent, the only way to get around the clutter is to interact with it using our meta-reasoner.

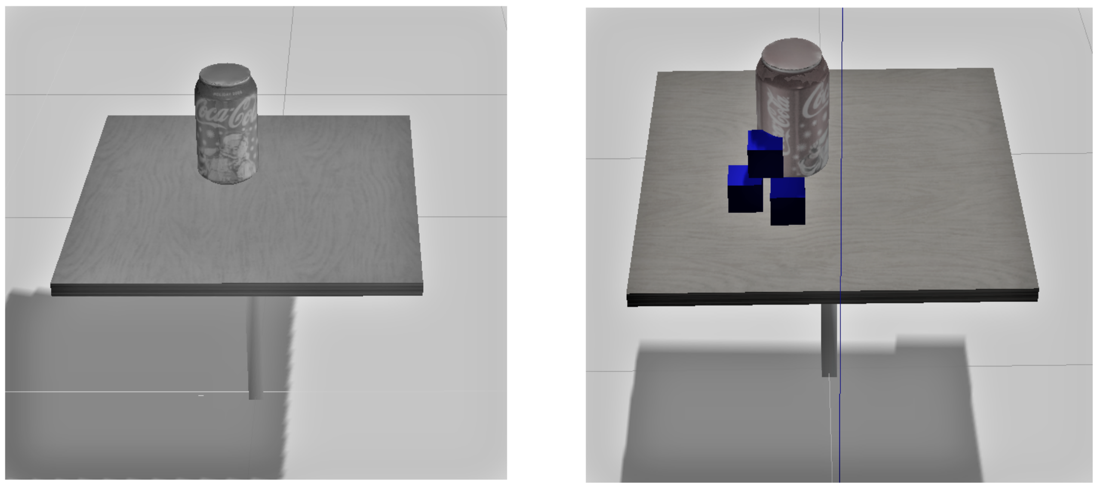
Figure 6 Original and variant grasping scenario in Gazebo

It is to be noted in this setup that both primitive actions are dependent on two other random variables, i.e., the choice of object, O_i, and placing position of the object in hand on the table, (x_{O_i}, y_{O_i}) respectively. Moreover, these two actions are also mutually exclusive, the arm cannot pick up another object before placing the object in hand somewhere, and an object cannot be placed if the gripper is empty. Therefore, the actual complexity lies in knowing which object to manipulate and at what position to place the object. Consecutively the learning in this experiment takes place for these two random variables as opposed to a. The input to reinforcement learner is given as a $(n + 2)$-tuple, where n is the total number of new objects in the scene. This consists of an n-tuple hot vector, $\hat{o} = \{o_1, o_2, \ldots, o_{n-1}\}$,where o_i is 1 if object O_i lies in collision with the padded path plan suggested by the original task plan. This hot vector is appended with two more categorical variables signifying the state of the robot namely, state of agent's gripper (open/close) and reference to object in the agent's hand. The configuration space of the planned path is thresholded at 1 foot or 0.3 meters around the path plan to account for control errors in robotic execution and compensate for the size of the gripper. The hyper-parameters used for this experiment:

- a starter ϵ of 1.0 with a $decay\_rate$ of 0.95 over 3000 iterations,

- a decaying learning rate starting at 1.0 following same decay as ϵ,
- and a greedy, decaying γ starting at 0.1 with a $decay\_rate$ of 0.93.

The table-top positions, in an ideal world, will be conditioned on the attributes of the object being placed. However, in our experiment all the novel objects are identical, i.e., cubes. Therefore, we have learned these positions as a Gaussian Mixture Model of the placements which lead to the emission of a positive reward. Position selection also follows the same strategy of epsilon-greedy as the object selection, i.e., either choose a position on table randomly with a probability of ϵ or randomly select from the rewarding poses stored in the GMM. The state representation being used for reinforcement learning affects the learning curve. Therefore, for the fairness of comparison, we have supplied the same reduced state to the pure learner as well as are using the same hierarchical learning setup, with the difference being that the combination learner receives the intermediate rewards for making the variant environment more like the original.

7 Experimental Results

We evaluate our results on two different aspects: the ability to completely solve the novel variant task, and its time efficiency as compared to a vanilla learner. The former is simply a score of whether the agent can reach the goal state or not. It is a binary score of 0 or 1.

The latter metric is to compare the efficiency of our proposed architecture versus one which cannot 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 variant configuration. Instead of reasoning about information gap and learning a bridging method, this pipeline follows a vanilla epsilon-greedy learning policy by employing a learner which learns an entirely new method from the 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.

7.1 Gold-hunting Scenario

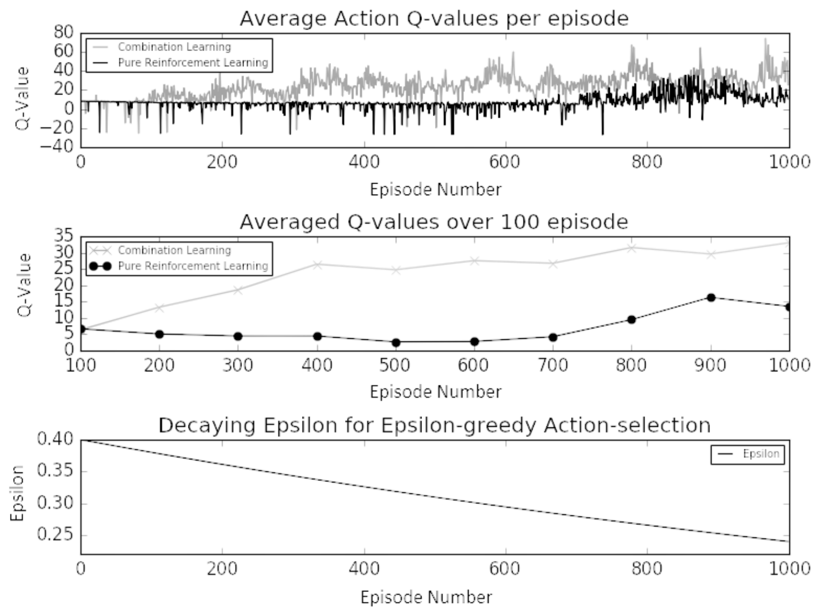

Figure 7 Comparing action Q-values for different approaches in Minecraft environment

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 combination learner is compared against the pure RL agent, the latter resulted in zero percentage of success in completing the mission over 1000 iterations. We hypothesize that the proposed scenario was a little too complicated for a simple algorithm like zero-order 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 vanilla RL with viable goals available and combination learner with viable goals as well as action cost discounts available.

As readers can see in Figure 7, the Q-values for pure reinforcement learning approach first takes a dip before gaining value. This is due to the agents repeatedly wrong actions which further decrease its confidence. The topmost plot shows a considerable number of spikes and jumping around for the Q-values, this is because the ϵ for our action-selection strategy is still high with the lowest value of 0.25. This leads to execution of random actions by the agent, but since our environments solution relying on a strictly sequential series of actions even one wrong random action can lead the agent down an action trajectory with zero reward returns. 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 value and to display the comparison more clearly.

7.2 *Grasping Scenario*

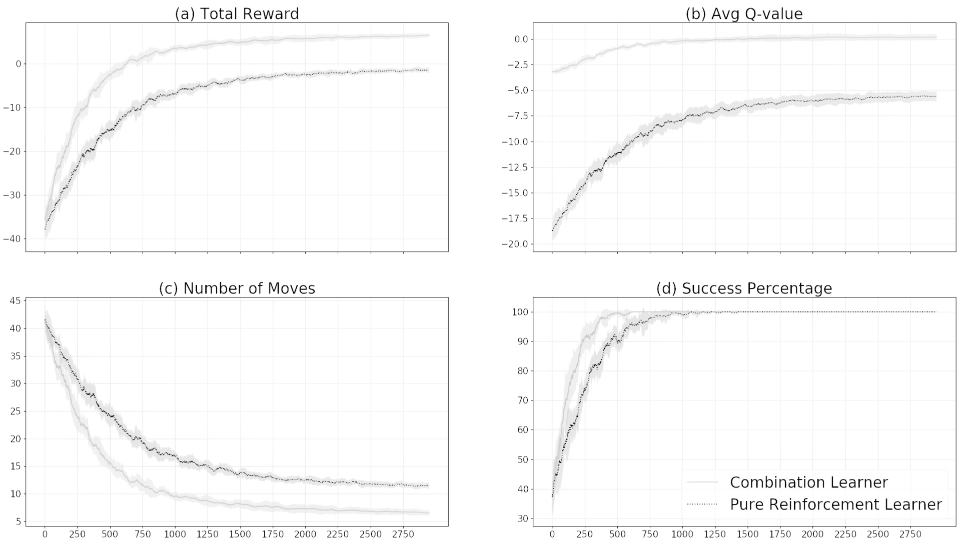

Figure 8 Comparison of pure-learner with combination-learner for Gazebo task

Figure 8 shows the comparative results over various aspects of the pure learner and combination learner. These results are solely for learning which objects to be picked up to clear the plan area and use a rolling mean on a window of 50 episodes to display a clearer picture. Not only does the combination learner stay ahead of the pure learner, but it does so by a margin of 750 episodes, regarding succeeding consistently (sub-figure 8.(d)). Moreover, as can be seen in sub-figure 8.(c), the combination learner can learn the most efficient way of solving this problem with a minimum number of moves which the pure learner does not approach even after 3000 episodes. The biggest reason for this difference is related to sub-figure 8.(b). The combination learner starts off with a comparatively confident estimate of Q-values which helps in shaping its exploration later by pushing it towards the more "fruitful" state-space. Figure 9 shows the learned heat-map for good placements of the picked object on the table such that the "plan area" remains cleared.

8 Discussion

We would like to begin by discussing some of the assumptions and limitations of our method. Our procedure makes strong assumptions about complete and noiseless observability, as well as the class of problems it is tackling. We still need to conduct some exploratory work to see how much of our framework can be applied to environments which change differently. Furthermore, basing discrepancies on a pixel-by-pixel or cell-by-cell comparison can be extreme for noisy sensors. It would be better if we can learn or reason for a baseline about how much dissimilarity is harmless.

Matrix computations are already costly, and when scaled up to a whole environment this comparison can be hard to handle for a resource-limited platform. Resource exhaustion is an issue that our usage of occupancy grids had the potential to run into, but the use of configuration spaces to prune down the sensor information to a much smaller volume and using grids to compress the data by discretization further helped immensely. Not only was this effect evident in computation time, but also concerning the learning curve since the state data was now based on a much smaller set of environment cells, which affects tabular Q-learning.

One of the critical insights about our algorithm stems from Figure 9, i.e., our approach does not improve much for problems where the goal is to learn the distribution of the value of a random variable. However, our approach evidentially holds merit for problems seeking a chain of actions as the solution, Figure 7 and Figure 8. Another interesting observation is illustrated in Figure 10 where our agent learns “how to not die” faster than the naive learner. Such an ability is essential for environments where the reward is delayed, and it is imperative for the agent to be active until then.

Lastly, we want to compare our method against some of the contemporary work. (Dannenhauer & Muñoz-Avila 2015*a*) prove that hierarchical expectations work better than flat plans and expectations for reasoning about goals; however, their semantic ontology of expectations are much more restrictive than our occupancy grids. Comparing our work to the KBRL spectrum, there are similar themes like reward shaping using a known plan (Grounds & Kudenko 2008, Grzes & Kudenko 2008), however, in our case, we dealt with a particular class of problems where apart from original plan the knowledge of how the environment has changed also affected the reward function. (Efthymiadis et al. 2016) look at a problem where stored, possibly incorrect, knowledge is revised for a specific task but they use a propositional form of expectations, and in contrast to their assumption of wrong information our system assumes known information is correct and focuses on learning new knowledge for overcoming the seen discrepancy. Additionally, our results reflect the findings of (Ulam et al. 2005, 2008) where they saw a considerable speed-up of the learning process by providing it with internal model and knowledge about the game world. However, our algorithm does not rely on as strict a definition of expectations and outlines a more general method of representing and calculating similarities at a lower-level, applicable to any map-based world.

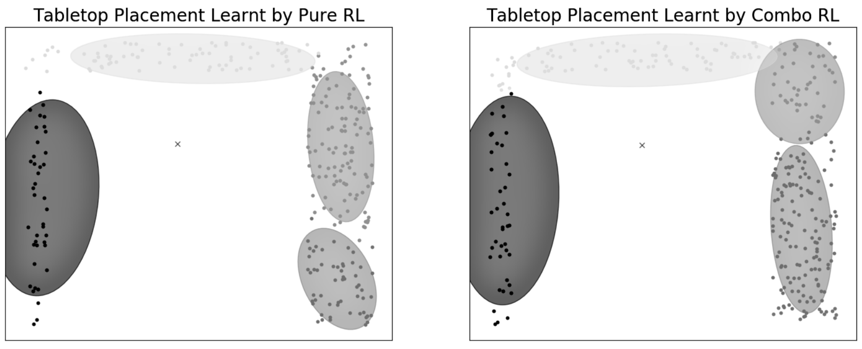

Figure 9 Learned distribution of good table-top placement locations (x marks the location of the can)

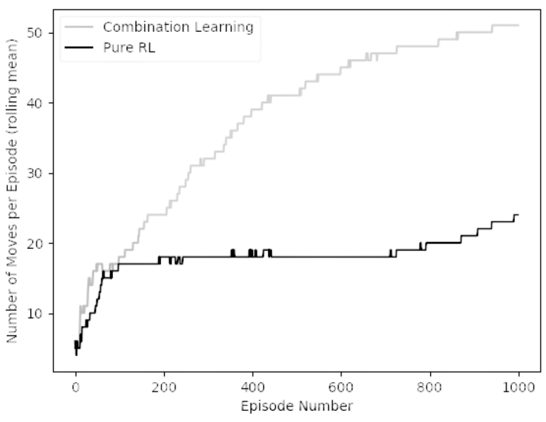

Figure 10 Number of Moves taken by Minecraft agent per episode

9 Conclusion

This paper presented a meta-reasoning architecture and approach for building AI agents situated in dynamic environments in which new objects with unknown action models are added. The architecture combines HTN planning and Q-learning within the meta-reasoning architecture. We performed experiments on two different tasks on two different platforms to test and evaluate our architecture. Our results suggest that the meta-reasoner can use the Q-leaner to complete the plans for the HTN planner in the new environment with added objects. This capability is potentially valuable because it enables us to have a flexible planner capable of extending its knowledge base to new worlds.

Moreover, we used low-level percepts in the form of occupancy grids for comparing plan expectations and actual observations, which makes our work suitable not only for software agents in simulated environments but potentially also for embodied agents that work on real raw data. Additionally, we showed that the guidance that the HTN provides to reinforcement learner helps in scoping the exploration of a large state-space by a significant factor. Such an architecture beats pure data-driven learners not only concerning learning efficiency but also flexibility since learned methods are grounded in states and can be strung together in any order to accomplish different tasks.

References

Anderson, M. L. & Oates, T. (2007), 'A review of recent research in metareasoning and metalearning', *AI Magazine* **28**(1), 12.

Argall, B. D., Chernova, S., Veloso, M. & Browning, B. (2009), 'A survey of robot learning from demonstration', *Rob. Auton. Syst.* **57**(5), 469–483.

Blum, A. L. & Furst, M. L. (1997), 'Fast planning through planning graph analysis', *Artificial Intelligence* **90**, 281–300.

Breazeal, C. (2004), *Designing Sociable Robots*, MIT Press.

Breazeal, C. & Scassellati, B. (2002), 'Robots that imitate humans', *Trends Cogn. Sci.* **6**(11), 481–487.

Cox, M. T. (2005), 'Field review: Metacognition in computation: A selected research review', *Artif. Intell.* **169**(2), 104–141.

Cox, M. T., Alavi, Z., Dannenhauer, D., Eyorokon, V., Muñoz-Avila, H. & Perlis, D. (2016), MIDCA: A metacognitive, integrated Dual-Cycle architecture for Self-Regulated autonomy, *in* 'AAAI', aaai.org, pp. 3712–3718.

Cox, M. T., Muñoz-Avila, H. & Bergmann, R. (2005), ‘Case-based planning’, *Knowl. Eng. Rev.* **20**(3), 283–287.

Cox, M. T. & Raja, A. (2011), *Metareasoning: Thinking about Thinking*, MIT Press.

Dannenhauer, D. & Muñoz-Avila, H. (2015*a*), Goal-Driven autonomy with Semantically-Annotated hierarchical cases, *in* E. Hüllermeier & M. Minor, eds, ‘Case-Based Reasoning Research and Development’, Vol. 9343 of *Lecture Notes in Computer Science*, Springer International Publishing, Cham, pp. 88–103.

Dannenhauer, D. & Muñoz-Avila, H. (2015*b*), ‘Raising expectations in GDA agents acting in dynamic environments’, *IJCAI* .

Dannenhauer, D., Muñoz-Avila, H. & Cox, M. T. (2016), Informed expectations to guide GDA agents in partially observable environments, *in* ‘IJCAI’, pp. 2493–2499.

Efthymiadis, K., Devlin, S. & Kudenko, D. (2016), ‘Overcoming incorrect knowledge in plan-based reward shaping’, *Knowl. Eng. Rev.* **31**(1), 31–43.

Efthymiadis, K. & Kudenko, D. (2013), Using plan-based reward shaping to learn strategies in StarCraft: Broodwar, *in* ‘2013 IEEE Conference on Computational Inteligence in Games (CIG)’, pp. 1–8.

Efthymiadis, K. & Kudenko, D. (2015), Knowledge revision for reinforcement learning with abstract MDPs, *in* ‘Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems’, AAMAS ’15, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp. 763–770.

Erol, K., Hendler, J. & Nau, D. S. (1994), HTN planning: Complexity and expressivity, *in* ‘AAAI’, Vol. 94, pp. 1123–1128.

Fitzgerald, T., Bullard, K., Thomaz, A. & Goel, A. K. (2016), Situated mapping for transfer learning, *in* ‘Fourth Annual Conference on Advances in Cognitive Systems’.

Goel, A. K. & Jones, J. (2011), Metareasoning for Self-Adaptation in intelligent agents, *in* ‘Metareasoning’, The MIT Press.

Goel, A. K. & Rugaber, S. (2017), ‘GAIA: A CAD-Like environment for designing Game-Playing agents’, *IEEE Intell. Syst.* **32**(3), 60–67.

Grounds, M. & Kudenko, D. (2008), Combining reinforcement learning with symbolic planning, *in* ‘Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning’, Springer Berlin Heidelberg, pp. 75–86.

Grzes, M. & Kudenko, D. (2008), Plan-based reward shaping for reinforcement learning, *in* ‘2008 4th International IEEE Conference Intelligent Systems’, Vol. 2, pp. 10–22–10–29.

Hammond, K. J. (2012), *Case-Based Planning: Viewing Planning as a Memory Task*, Elsevier.

Jaidee, U., Muñoz-Avila, H. & Aha, D. W. (2011*a*), ‘Case-based learning in goal-driven autonomy agents for real-time strategy combat tasks’, *Proceedings of the ICCBR* .

Jaidee, U., Muñoz-Avila, H. & Aha, D. W. (2011*b*), Integrated learning for goal-driven autonomy, *in* ‘IJCAI Proceedings-International Joint Conference on Artificial Intelligence’, Vol. 22, p. 2450.

Jaidee, U., Muñoz-Avila, H. & Aha, D. W. (2012), Learning and reusing Goal-Specific policies for Goal-Driven autonomy, *in* B. D. Agudo & I. Watson, eds, ‘Case-Based Reasoning Research and Development’, Vol. 7466 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 182–195.

Jones, J. K. & Goel, A. K. (2012), ‘Perceptually grounded self-diagnosis and self-repair of domain knowledge’, *Knowledge-Based Systems* **27**, 281–301.

Koenig, N. & Howard, A. (2004), Design and use paradigms for gazebo, an open-source multi-robot simulator, *in* ‘2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)’, Vol. 3, pp. 2149–2154 vol.3.

Kolodner, J. (2014), *Case-Based Reasoning*, Morgan Kaufmann.

Leake, D. B. (1996), *Case-Based Reasoning: Experiences, Lessons and Future Directions*, 1st edn, MIT Press, Cambridge, MA, USA.

Lozano-Perez, T. (1983), ‘Spatial planning: A configuration space approach’, *IEEE Trans. Comput.* **C-32**(2), 108–120.

Muñoz-Avila, H., Jaidee, U., Aha, D. W. & Carter, E. (2010), Goal-Driven autonomy with Case-Based reasoning, *in* ‘Case-Based Reasoning. Research and Development’, Springer Berlin Heidelberg, pp. 228–241.

Murdock, J. W. & Goel, A. K. (2001), Meta-Case-Based reasoning: Using functional models to adapt Case-Based agents, *in* ‘Case-Based Reasoning Research and Development’, Springer Berlin Heidelberg, pp. 407–421.

Murdock, J. W. & Goel, A. K. (2008), ‘Meta-case-based reasoning: self-improvement through self-understanding’, *J. Exp. Theor. Artif. Intell.* **20**(1), 1–36.

Murdock, J. W. & Goel, A. K. (2011), *Self-Improvement through Self-Understanding: Model-Based Reflection for Agent Self-Adaptation*, Amazon.

Nau, D. S., Au, T. C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D. & Yaman, F. (2003), ‘SHOP2: An HTN planning system’, *1* **20**, 379–404.

Nau, D. S., Cao, Y., Lotem, A. & Munoz-Avila, H. (1999), SHOP: Simple hierarchical ordered planner, *in* ‘Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2’, IJCAI’99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 968–973.

Ng, A. Y., Harada, D. & Russell, S. (1999), Policy invariance under reward transformations: Theory and application to reward shaping, *in* ‘ICML’, Vol. 99, pp. 278–287.

Nilsson, N. J. (1998), *Artificial Intelligence: A New Synthesis*, Elsevier.

Nilsson, N. J. (2014), *Principles of Artificial Intelligence*, Morgan Kaufmann.

Ontanón, S., Mishra, K., Sugandh, N. & Ram, A. (2010), ‘On-line case-based planning’, *Comput. Intell.* **26**(1), 84–119.

Paisner, M., Maynord, M., Cox, M. T. & Perlis, D. (2013), Goal-driven autonomy in dynamic environments, *in* ‘Goal Reasoning: Papers from the ACS Workshop’, p. 79.

Riesbeck, C. K. & Schank, R. C. (2013), *Inside Case-Based Reasoning*, Psychology Press.

Stroulia, E. & Goel, A. K. (1995), ‘FUNCTIONAL REPRESENTATION AND REASONING FOR REFLECTIVE SYSTEMS’, *Appl. Artif. Intell.* **9**(1), 101–124.

Stroulia, E. & Goel, A. K. (1999), ‘Evaluating PSMs in evolutionary design: the a UTOGNOSTIC experiments’, *Int. J. Hum. Comput. Stud.* **51**(4), 825–847.

Sutton, R. S. & Barto, A. G. (1998), *Reinforcement learning: An introduction*, Vol. 1, MIT Press Cambridge.

Thrun, S., Burgard, W. & Fox, D. (2005), *Probabilistic Robotics*, MIT Press.

Ulam, P., Goel, A. K., Jones, J. & Murdock, W. (2005), 'Using model-based reflection to guide reinforcement learning', *Reasoning, Representation, and Learning in Computer Games* p. 107.

Ulam, P., Jones, J. & Goel, A. K. (2008), Combining Model-Based Meta-Reasoning and reinforcement learning for adapting Game-Playing agents, *in* 'AIIDE'.

Vattam, S., Klenk, M., Molineaux, M. & Aha, D. W. (2013), Breadth of approaches to goal reasoning: A research survey, Technical report, Naval Research Lab Washington DC.

Watkins, C. J. C. H. & Dayan, P. (1992), 'Q-learning', *Mach. Learn.* **8**(3), 279–292.

Appendix A Algorithms

```
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
```

Figure A.1 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
```

Figure A.2 Meta-reasoner Algorithm