CHAPTER

2

Analogy and metareasoning: Cognitive strategies for robot learning

Ashok K. Goel[a], *Tesca Fitzgerald*[a], *Priyam Parashar*[b]

[a]Georgia Institute of Technology, Atlanta, GA, United States [b]Contextual Robotics Institute, UC San Diego, La Jolla, CA, United States

2.1 Background, motivations, and goals

All intelligent forms share a conundrum: they not only must deal with novel situations but also must start with what they already know; how, then, can any intelligent form deal with any novelty? Of course, no intelligent form addresses all novel situations very well; if and when it does not, it may be unable to accomplish its goals or suffer harm in the process, perhaps even perish. Thus, if we are to build long-living robots, we must provide them with the capability of dealing with at least some kinds of novelty. Further, if we want robots to live and work among humans, then we may start with humanlike, human-level, *cognitive* strategies to addressing novel situations.

Research on cognitive science suggests four broad categories of mechanisms for addressing novel situations. The first is situated cognition and learning. The intelligent agent, human or robot, starts with knowledge of the world in the form of a policy of actions to be taken in different states in the world, executes the policy, often fails, but learns from the failure by incrementally modifying the policy. The mechanism of situated (or reinforcement) learning not only is general purpose and can be quite effective but also can be computationally costly.

The second cognitive mechanism is through social learning. The agent starts with knowledge of primitive objects, relations, events, and actions. The intelligent agent is situated in a sociocultural context where the agent may learn by observing a teacher, or the teacher may actively help the agent learn through demonstrations or corrections, or the agent may actively learn by asking questions of the teacher. This interactive mechanism can be costly, too, when a large number of observations and demonstrations are needed, but if a faithful teacher who patiently provides trustworthy answers is available, then this method can be quite effective.

https://doi.org/10.1016/B978-0-12-820543-3.00002-X

The third cognitive mechanism for dealing with novelty is by analogy. Here the intelligent agent starts with a memory of previously encountered source cases. Given a new target problem, the agent retrieves a relevant case from memory and transfers and adapts knowledge from the known case to the new problem. The emphasis here is on transfer of relationships among objects in the known case and in the new problem, not necessarily on the objects themselves. This mechanism assumes a memory of previously encountered cases, but it can be quite effective when a case relevant to the new problem can be found in memory.

Finally a fourth cognitive mechanism for addressing novelty is through metareasoning—reasoning about reasoning. Here the agent starts not only with knowledge about the world but also knowledge about itself in the form of a model of its knowledge and reasoning. When the agent encounters a new problem, it uses its knowledge to address the problem and fails but collects additional information through the failure. It now uses metareasoning to localize and repair the causes of failures and thus incrementally adapts itself to the novel situation. This method requires a self-model and can be computationally costly for small problems but can be quite effective for complex problems with interacting goals.

We note the importance of context in this discussion. The choice of a specific mechanism among the aforementioned four categories depends very much on both the external context in which the agent is situated (is a teacher available to show a demonstration or address a question?) and the internal context of the agent's knowledge (is a case relevant to the new situation available?). We also note that the four mechanisms are quite complementary to one another, not mutually exclusive. In fact, human cognition appears to select, integrate, shift, and abandon among the various mechanisms depending on the constantly changing external (the novelty in the world) and internal (knowledge and processing) contexts.

These four cognitive strategies provide useful starting points for designing long-living robots capable of addressing novelty. In this chapter, we describe two combinations of the aforementioned cognitive strategies for dealing with novelty in the context of interactive robotics. The first approach combines the strategies of social learning with analogical reasoning (Fitzgerald, Goel, & Thomaz, 2018). A teacher first demonstrates a skill to a robot, and the robot stores the learned knowledge as a case. Given a new situation the robot analogically transfers the skill from the case to the new situation. The second approach combines reinforcement learning and metareasoning (Parashar, Goel, Sheneman, & Christensen, 2018). The agent uses deliberative planning to achieve a familiar goal in a novel situation. If and when it fails, it uses local reinforcement learning to learn the correct action in the failed state. A metareasoner monitors and coordinates the external behaviors and the internal processing in the agent. We will conclude this chapter with a discussion of the implications of this work in robotics on cognitive theories of analogy and metareasoning.

2.2 Using social learning and analogical reasoning in cognitive robotics

Let us consider the situation illustrated in Fig. 2.1. First a human teacher demonstrates a skill to accomplish a task to a robot (left of the figure). Then the teacher asks the robot to address the same task in a new but similar situation in which some of the objects and object relationships are different (right of the figure). How might the robot transfer the skill acquired in the prior demonstration to the new situation?

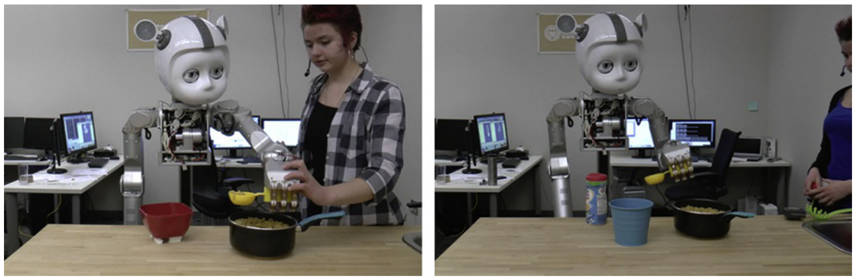

FIG. 2.1 A human teacher demonstrates how to scoop pasta from a pot into a bowl (as shown in the figure on the *left*). The robot is then asked to scoop pasta from a different pot into a different bowl (shown on the *right*). The question becomes how may the robot not only build an analogical mapping between the two situations but also transform the action model from the first situation—how to grasp the scoop, how to move it in space, how to dip it into the pot, etc.—to the new situation.

In traditional approaches to this problem, also called "transfer learning," the teacher gives a large number of demonstrations of the skill, the robot learns a model from the demonstrations, and then, given the new situation, applies the model. However, the cognitive strategy of analogical reasoning suggests the robot may build a mapping between the demonstrated source case and the target new problem and transfer the skill learned in the demonstration to the new situation. In principle, this strategy requires much fewer initial demonstrations.

While analogical mapping has been addressed in related work, it is typically assumed that the robot can gain additional experience in the target environment or knows which object feature(s) to use when evaluating object correspondences. However, when the robot learns a task from human interaction, such assumptions lead to two challenges. First, gaining additional experience in the target environment can be a time-consuming task in which the human teacher must continue to provide task demonstrations or feedback to the robot as it explores the environment. Simply helping the robot gain the correct mapping would more efficiently utilize the teacher's time. Second the robot does not have the same contextual knowledge that the human teacher has about the task and thus does not know which object features are relevant to successfully repeating the task. For example, in transferring a cup-stacking task, the robot would need to identify which cups in the source and target environments are similar according to their size feature. In another task in which the robot learns to sort the same cups by their color, the robot would need to instead map objects according to their hue feature. While the human teacher knows which features are relevant in the context of that task, the robot does not.

In these problems, which we call "situated mapping" problems, the robot must identify the mapping that maximizes similarity between source environment objects and their equivalent objects in the target environment. However, defining this similarity metric (and the object features it considers relevant) is nontrivial. The similarity metric that is appropriate for one task may not represent the object features relevant to another task. Consider our previous example, where a robot learns to use cups in both a stacking task and a sort-by-color task. In the first task the robot should identify an object mapping maximizing the similarity between mapped objects based on their size feature, whereas the second task requires objects to be mapped according to their hue feature.

We address this problem of context-dependent mapping, in which object mapping is dependent on the task being performed. The interactive object mapping problem has the following stages of interaction with a human teacher: The human teacher demonstrates a task in the source environment. The robot observes the object features and task steps involving those objects. The robot observes the target environment containing new objects and is asked to repeat the newly learned task. The robot infers the mapping between objects in the source and target environments. The robot uses this mapping to transfer the learned task for execution in the target environment. Our work is primarily concerned with how additional interaction between the robot and the human teacher can facilitate step 4: inferring the object mapping.

2.2.1 Related work

Research on cognitive systems has long addressed object mapping, particularly by identifying structural similarity between source and target objects, that is, objects sharing similar relations to their surroundings (Gentner, 1983; Gentner & Markman, 2006; Gick & Holyoak, 1983). The representation of the relationship between objects informs the mapping process. Falkenhainer, Forbus, and Gentner (1989) define a scene as a graph; objects sharing similar structural relations within their own graphs are mapped. Alternatively, objects may be represented and compared in terms of their pragmatic, spatial, and semantic features (Holyoak & Thagard, 1989) or in terms of visual analogies (Davies, Goel, & Yaner, 2008). While these approaches have been demonstrated in simulated domains, they have not been applied to identify analogies between physical objects with a rich set of perceptual features.

Learning from demonstration enables a robot to quickly learn skills from a human teacher (Akgun et al., 2012; Argall et al., 2009; Chernova & Thomaz, 2014). Prior work has demonstrated how a robot may ground an abstract task representation, such as "task recipes" (Misra et al., 2016; Waibel et al., 2011), in semantic locations in a new environment via interactive demonstrations (Bullard et al., 2016), learning prototypical object usage (Tenorth, Nyga, & Beetz, 2010), and learning to classify objects via active learning (Kulick et al., 2013). While related to object mapping, the symbol grounding problem differs in that it grounds abstract object references in specific object perception (whereas mapping directly correlates two object instances).

Similar mapping problems have also been explored in reinforcement learning domains; by learning an intertask mapping between source and target state variables, an agent can transfer a learned policy to a new domain (Taylor, Stone, & Liu, 2007). While a robot can build these mapping by exploring the target environment, we aim to minimize the need for data collection in the target environment.

Overall, current approaches to object mapping assume (i) objects with the same classifications play the same role in any task, (ii) the robot knows a priori which object features to use for mapping, (iii) the robot may continue to explore/train in the target environment, and/or (iv) an abstraction of the object can be used to identify specific instances of that object symbol.

However, these assumptions do not hold in situated mapping problems, where we would like the robot to receive very limited new data/demonstrations of a task and then identify an object mapping without knowledge of the specific object features relevant to that task.

2.2.2 Approach

We have presented the mapping-by-demonstration (MbD) approach to situated mapping, in which an agent uses mapping "hints," a limited number of object correspondences, to infer the remainder of the mapping (Fitzgerald, Bullard, Thomaz, & Goel, 2016). Since the human teacher is aware of both (i) the goal of the task and (ii) the role that each object plays in achieving that goal, we have proposed a human-guided object mapping method Fig. 2.2, consisting of two interaction phases and two mapping phases. The interaction phases include a demonstration phase in which the robot learns the task steps from demonstration and an assistance phase in which the robot records interactive assistance from the teacher. The assistance is used in the two mapping phases: first in a mapping inference phase and then in the confidence evaluation phase that determines whether additional assistance should be requested. We now describe all four phases, later evaluating phases in isolation via simulated, interactive, and offline evaluations.

2.2.2.1 *Demonstration phase*

At the start of the interaction, the robot observes the source environment using an RGB-D (color + depth) camera and extracting the location, size, and hue features of each object, after which it derives the set of spatial relations between each object and obtains affordance data (the actions enabled by that object, e.g., openable and pourable) and property data (variables associated with an object's affordances, e.g., an openable object that has the property of being open or closed). After the robot observes objects in its environment, the human teacher interacts with the robot's arm to physically guide it through executing the task (e.g., Fig. 2.3). Following the demonstration the task is represented as (i) the list of object representations and (ii) a list of task steps that indicate the primary object for each step.

2.2.2.2 *Assistance phase*

After the task demonstration the robot may receive assistance from the human teacher to repeat the task in the target environment, later using this assistance during the mapping inference phase. Having the teacher provide assistance via natural interaction (e.g., pointing at or picking up an object) mitigates the need for human teachers to have knowledge of which

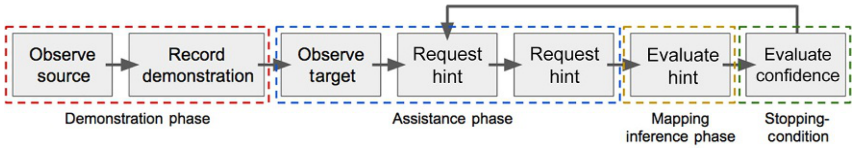

FIG. 2.2 We present mapping by demonstration, which consists of four phases: demonstration, assistance, mapping inference, and confidence evaluation.

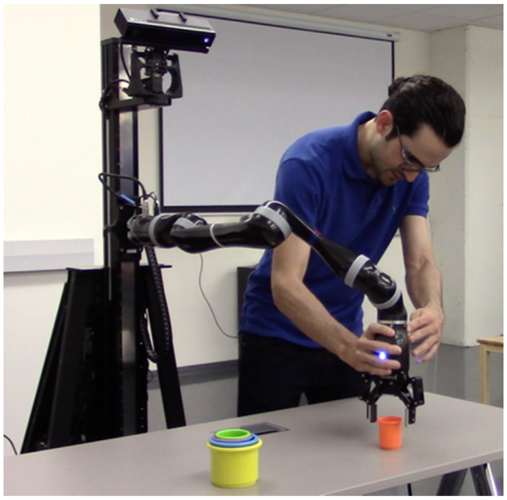

FIG. 2.3 A user provides a demonstration of a cup-stacking task by guiding the robot's end effector to complete the task.

features the robot has the capacity to observe (e.g., the robot can record object color, but not product brand) or how to express feature values (e.g., hue values).

Once the robot requests assistance (e.g., "Where do I go next?"), the human teacher provides a mapping assist by handing the robot the next object it should use to complete the task in the target environment (e.g., the interaction in Fig. 2.4A) or, if the robot is already holding an object, by pointing to where the robot should place the object in the target environment (e.g., Fig. 2.4B). Each mapping assist indicates a correspondence between (i) the object referenced by the teacher in the target environment and (ii) the object that would have been used in the next step in the original task demonstration.

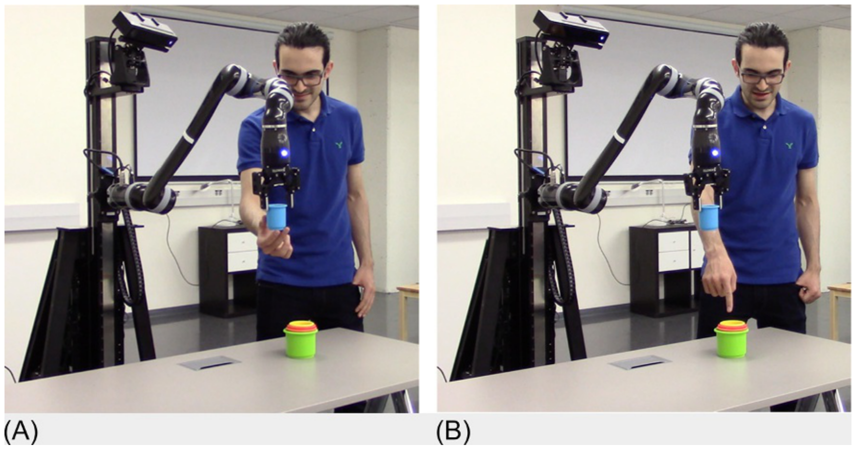

FIG. 2.4 (A) The teacher hands the robot the next cup it should stack. (B) The teacher indicates where the robot should place the cup it is currently holding.

2.2.2.3 *Mapping inference phase*

Two goals must be addressed simultaneously to infer the object mapping: selection of (i) an object similarity function (and associated object feature) that is representative of the mapping assistance received thus far and (ii) an object mapping that maximizes object similarity according to the selected similarity function. As such, our algorithm maintains both a (i) feature set space containing all feature sets under consideration for the similarity metric and (ii) a mapping hypothesis space containing all mapping hypotheses still under consideration.

The mapping hypothesis space is initialized as the set of all possible object mappings and thus begins as an $n!$-sized set for an n-to-n mapping problem. An $n \times n \times 7$ evaluation matrix is generated during initialization, containing the evaluation score for every possible object correspondence according to each of the seven-object features. The evaluation matrix only needs to be generated once (during initialization). Afterward the evaluation matrix is referenced to evaluate a mapping hypothesis. Many tasks may require additional features that have not been addressed here. However, this method is intended to consider a variety of features for object mapping and is still applicable in such tasks. Additional object features can be incorporated by defining a new evaluation metric and expanding the evaluation matrix to include the new metric.

A mapping assist consists of an object in the source environment and its corresponding object in the target environment. After initialization, each mapping assist is used to (i) prune the mapping hypothesis space, (ii) prune the feature set space, and then (iii) select the highest-ranked mapping hypothesis as the predicted mapping. Each combination of a mapping hypothesis and feature is then evaluated, and the highest-ranked mapping and feature set combination is then returned as the predicted mapping.

A single mapping assist may not provide enough information to infer the correct object mapping. Thus, after receiving a mapping assist and inferring the object mapping, a decision must be made to either request additional assistance or to complete the rest of the task autonomously using the most recently inferred object mapping. Similar to the confidence-based autonomy (CBA) approach introduced by Chernova and Veloso (2007, 2009), our work aims to enable the robot to rely on confidence as a means to managing its assistance from the human teacher. However, since the robot does not know which features are relevant to the task, it is unable to select features to calculate its confidence in the same manner as CBA. We propose a variation of confident execution that utilizes the information available during interactive object mapping: the evaluation matrix calculated during MbD to determine the confidence of its predicted mapping. We define confidence as the decision margin between these two top-ranked mapping hypotheses evaluation scores.

2.2.3 First experiment

We performed an extensive evaluation of the mapping inference phase in simulation using data from the ground-truth mapping as the source of mapping assistance. We evaluated the system with three categories of simulated tasks, containing 5–7 objects in the source and target environments. These categories represent incrementally more difficult problems; as the number of objects increases, the mapping hypothesis space from which the system must

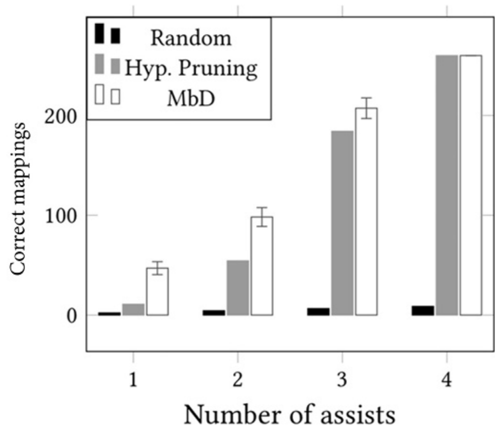

FIG. 2.5 Performance in five-object tasks.

choose a single mapping increases factorially. In total the simulated evaluation consisted of 780 evaluations, each performed for all permutations of mapping assist orderings.

Figs. 2.5–2.7 compare the performance of the MbD algorithm over all assistance orderings, with error bars denoting one standard deviation, for two baselines: (i) expected performance when selecting a random mapping without utilizing mapping assistance and (ii) expected performance when using mapping assistance to only prune the hypothesis space and then choosing a random mapping from the remaining hypothesis space (rather than using the assistance to infer features on which mapping is based).

2.2.3.1 *Discussion*

In any mapping problem the number of possible object mappings increases factorially with the number of objects present. Graph isomorphism is an intractable problem in general, and thus this attribute is inherent to any technique for object mapping. This problem motivates situating mapping in the task environment. While all mapping hypotheses are considered,

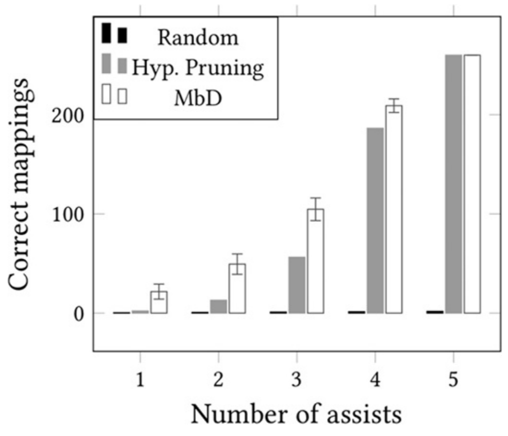

FIG. 2.6 Performance in six-object tasks.

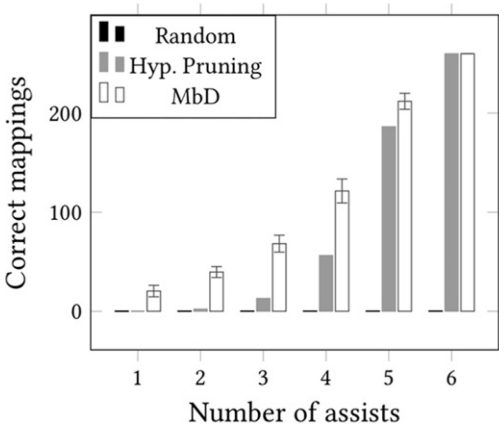

FIG. 2.7 Performance in seven-object tasks.

leveraging mapping assistance helps (i) prune the hypothesis space after each assist, (ii) prune the feature set space, and (iii) reevaluate the remaining mapping hypotheses to produce a mapping prediction.

The results indicate that using mapping assistance increases the robot's likelihood of predicting a correct mapping quickly. Even when mapping assistance is only used to prune the hypothesis space, as shown in the hypothesis pruning results in the simulated evaluation ("Hyp. Pruning" in Figs. 2.5–2.7), the robot's likelihood of selecting a correct mapping is dramatically increased over that of choosing an object mapping at random. The MbD algorithm provides further benefit by inferring additional information from the assistance; rather than only use the mapping assist to prune the hypothesis space, it is also used to infer the feature(s) on which mapping may be performed. This benefit is especially evident as the hypothesis space increases. Particularly, in the seven-object task (the category of mapping problems with the largest hypothesis space), the MbD algorithm correctly solves significantly more problems within the first 1–4 assists than either the hypothesis pruning or random mapping baselines.

These results are obtained using simulated assistance and thus rely on the assumption that mapping assistance is always correct. In a realistic interaction, however, the robot would need to obtain this assistance from the human teacher. This introduces several potential sources for error, such as misinterpretation of the human teacher's assistance or the teacher's misinterpretation of the task.

2.2.4 Summary

Without contextual knowledge about the task, the robot cannot weigh object features to identify an object mapping for task transfer. Prior work assumes that (i) the robot has access to multiple new demonstrations of the task or that (ii) the primary features for object mapping have been specified. Our method does not make either assumption; rather than requiring additional demonstrations of the task, it uses limited structured interaction with a human teacher. Additionally, by having human teachers provide mapping assistance by indicating objects (rather than describing features), we mitigate the need for teachers to have knowledge of which features the robot can observe or how to express feature values. More generally, these issues arise because of the grounding of analogical reasoning in perception and action.

2.3 Using reinforcement learning and metareasoning in cognitive robotics

To ground some of the issues and concepts involved, consider the example of a rational mobile agent operating in a Minecraft environment called Moon (Fig. 2.8). Moon is a virtually embodied agent, that is, it interacts with the various objects in a Minecraft environment under constraints of certain predefined models. Moon is outfitted with a planner, a knowledge base about the primitive actions of the agent, and the action models of the expected objects in the environment and a sensor to observe the environment. The action model of an object defines its behavior when a primitive action is applied to it. Moon is also optimistic in its behavior; it assumes each action will be successful as planned. In the current story, Moon is spawned in a two-room environment that is joined via an open door. Moon's aim is to gather gold blocks placed somewhere in room 2, that is, the room farthest from its spawn location.

In the current experiment, 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 for reuse. Moon also records the state transitions encountered to make sure a plan is reused as expected. However, Minecraft is a changing world. Imagine there lives a builder who 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 announcing every change. So now, unaware of these changes, Moon

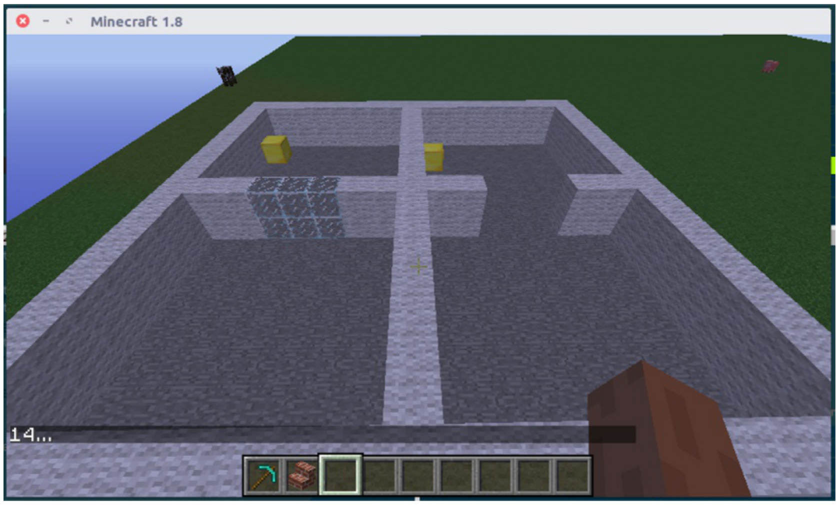

FIG. 2.8 Variant (*l*) and original (*r*) tasks in Minecraft.

starts its task of finding the gold block. As expected the unforeseen glass wall blocks the transition of an agent moving from room 1 to room 2, failing the plan.

Since glass was not expected, Moon does not have any knowledge of the material or how to interact with it. Several questions arise here. How can the agent describe a new artifact to the planner? Once defined, can we reuse some of the existing knowledge about the environment and the task to learn a policy from a current state to achieve the desired goal? And, finally, how should the policy be stored within the framework for future reuse?

2.3.1 Related work

Our work builds on the Autognostic, REM,[a] Augur, and Game Agent Interactive Adaptation (GAIA) projects. The Autognostic project (Stroulia & Goel, 1995, 1999) developed a multilayered agent architecture including situated, deliberative, and metareasoning components. The Autognostic system models an intelligent agent as a functional representation of internal processes, using the explicit modeling to repair its deliberative navigational planner. 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 (TMKL), more expressive than HTNs,[b] to capture functional models of agent design. REM addresses both retrospective and proactive adaptations. It uses metacase-based reasoning to proactively adapt for functionally similar tasks and using functional descriptions to retroactively localize failures. Unlike the Autognostic and REM projects, the Augur project (Goel & Jones, 2011; Jones & Goel, 2012) focuses on the use of metareasoning to diagnose and repair domain knowledge grounded in perception. Finally the GAIA[c] project (Goel & Rugaber, 2017) provides an interactive CAD-like environment for constructing game-playing agents (for playing Freeciv[d]) in the REM architecture and the TMKL2 language. The GAIA system enables interactive diagnosis and repair of the agents' design in case of failures.

This present work builds on the aforementioned research and especially the REM architecture and its combination of situated, deliberative, and metareasoning strategies. However, unlike previous REM research that focused on variations of tasks assigned to an intelligent agent 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.

Lastly, we want to compare our method against some of the contemporary work that integrates knowledge-based frameworks with reinforcement learning (Sutton & Barto, 1998).

[a] Reflective Evolutionary Mind, http://bill.murdocks.org/rem/.

[b] Hierarchical Task Networks, touched upon in Section 2.3.2.

[c] http://dilab.gatech.edu/publications/gaia-a-cad-like-environment-for-designing-game-playing-agents/.

[d] An open-source empire-building strategy game, http://www.freeciv.org/.

Dannenhauer and Muñoz-Avila (2015) prove that hierarchical expectations work better than flat plans and expectations for reasoning about goals; however, their semantic ontology of expectations is much more restrictive than our occupancy grids, which is a discretized representation of 3D space explained more deeply in the next section. Additionally, our results reflect the findings of Ulam, Goel, Jones, and Murdock (2005) and Ulam, Jones, and Goel (2008) where they saw a considerable speedup of the learning process by providing it with an 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.

2.3.2 Approach

In this section, we will first describe the overall architecture and the intuitions of the system before outlining the details of the components used. The architecture is designed to allow the processes of metareasoning about an agent's plan execution and gives an overview of the communications between the different components.

2.3.2.1 Architecture and concepts

The architecture is organized into three layers of reasoning: (a) situated or object-level reasoning that acts and observes directly interfacing with the environment; (b) deliberative reasoning that reasons about object-level information and acts to construct a plan; and finally (c) a metareasoning level that consumes both object-level and deliberative-level information, reasons for choosing the goals for a situated learner based on the tasks and observations from the past, and acts to switch control between deliberative reasoning and situated learning (Fig. 2.9).

2.3.2.2 Object layer

This layer consists of motion-inducing actuators, environment encoding sensors, the environment itself (extracted and implemented using the simulation engine and APIs), and a reactive tabular Q-learner (Fig. 2.13). The actuators induce behavioral or physical changes in the agent situated in the environment that reflect the function of its primitive actions. The primitive actions are domain and agent dependent and exhibit an indirect form of knowledge encoding. The environment is observed using a laser scanner sensor that emits an array of equally distanced laser rays at discretized heights into the environment and returns the environment information as an observation array of the distance at which each ray reflects off of an environmental artifact (infinity if it does not) and the reflected value or percentage of the ray intensity that indirectly encodes the material hit by the ray. This 3D array of discretized environment encoding is called an occupancy grid, as it describes the occupancy of each world cell. The reason for this representation choice is twofold: (a) Occupancy grids are the data structures of choice in the 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 that accounts for the physical properties makes more sense than using any higher-level descriptors. Occupancy grids in both the experiments in current work correspond to a 270-degree sweep of the environment centered on the agent. Each cell in the occupancy grid is a data structure with two elements. The occupancy variable can have one of the three possible values, FREE, OCCUPIED, or UNKNOWN, and the intensity variable stores the reflected reading of the ray. Like primitive actions the height up until which the scan reading

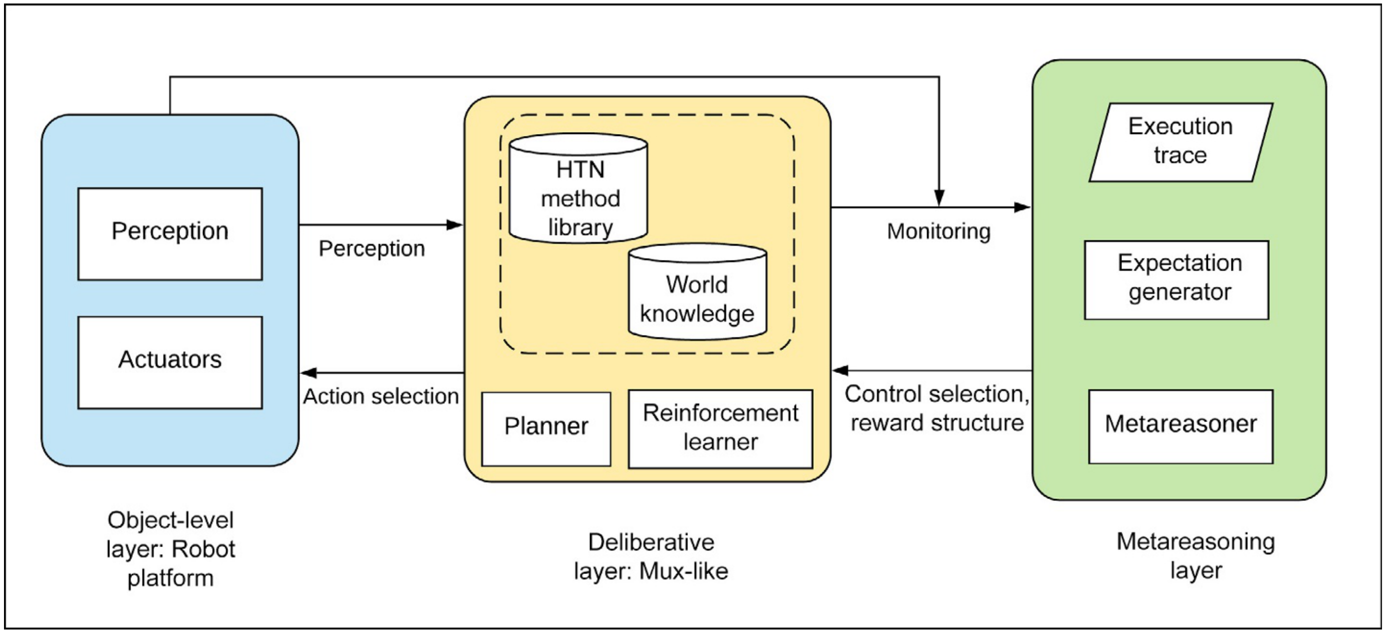

FIG. 2.9 Proposed system architecture.

should be considered for planning depends on the configuration space (Lozano-Perez, 1983) of the agent. This is a robotics concept used to calculate the volume of interest that should be considered for avoiding obstacles. Usually for planning the configuration space is created by padding obstacles to find a safe plan, as in our case; however, since we already have the path, we all padded space around it to represent the physical body of the robot and consider this as the volume of interest for metareasoning processes.

2.3.2.3 *Deliberative layer*

The deliberative layer consists of the main planner that is instantiated as a hierarchical task network planner (Erol, Hendler, & Nau, 1994). The planner itself is an off-the-shelf Python[e] version of SHOP[f] (Nau, Cao, Lotem, & Munoz-Avila, 1999) algorithm that grounds concepts like preconditions, effects, subtasks, and state descriptions into Python descriptions and objects. This layer also houses the method and subtask library associated with the planner. This layer takes as input the goal state of the task to be solved and returns a hierarchical sequenced array of methods and a corresponding sequence of primitive actions to be executed at the object level. Additionally, the *trace memory*, that is, the state transitions logged by Moon in Section 2.3, is also situated in this layer. The trace consists of the object-level observations and the sensor encoding of the environment in the form of 3D occupancy grids. To incorporate this data the HTN formalism was modified to include this step as execution metadata.

2.3.2.4 *Metareasoning: The processes*

This layer has bidirectional communication channels to both the object-level and deliberative layer. It uses the traces stored at the deliberative layer to monitor the developments at the object level. This monitoring is done by comparing the occupancy grid at the end of a method execution from the past to the current occupancy grid observed at the end of the same method. The comparison is done on an element-by-element basis between the two arrays; the comparison metric is designed to take into account the relative ordering of the observations.

A discrepancy between the expected and observed is defined by the dissimilarity score of the corresponding occupancy grids and a list of indices where the grid readings do not match. This layer also houses the control switch that invokes the planner or the reactive Q-learner with a goal.

2.3.3 Algorithm

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 environment state variant, then it is retrieved for execution while the associated metadata, that is, the abstracted HTN method sequence and corresponding trace, is sent to the metareasoner for monitoring. Given the dissimilarity score generated using the procedure in Section 2.3.2.2, a discrepancy report is generated that includes the score, a list of the grid cell indices that contributed, and the precondition of all methods in the partial HTN plan after the current method. Current implementation uses a conservative threshold of 0 that means, if an execution state transition is different from the expectation by even one cell, then it is noted as a discrepancy.

[e] https://bitbucket.org/dananau/pyhop.

[f] Simple Hierarchical Ordered Planner.

This procedure works well for our controlled simulated environments but should be calibrated for disregarding ambient noise in dynamic environments. In current work, we specifically consider the class of discrepancy that occurs due to the addition of new objects and artifacts to the environment, therefore, our choice of the dissimilarity computation procedure.

Next in phase two, or the *assess* process, the metareasoner sets up a knowledge-informed learning problem for the tabular Q-learner to solve. We use the partial plan returned with the discrepancy report to do two things: (a) The preconditions of postdiscrepancy methods are used as alternative and viable goals for the Q-learner, which helps set up a more dense reward system; and (b) the lower-level expectations are compared at each Q-learning cycle to provide additional reward to the learner, the intuition being that the more the agent makes the current situation like the last one, the more possible it is to solve it like the previously known task situation. For the latter end the metareasoner constructs a reward strategy where the agent gets a random discount (between 0 and 0.5) for actions that make the current environment more like the previously known one. This reward layer also indirectly encourages an exploration of the environment. This reward is tracked by noting a decrease in the dissimilarity score between the current and prior observations. Phase three is the *guide* process where this constructed problem is passed to the Q-learner, and the metareasoner falls back waiting on a return from the object-level layer.

Phase four is marked by a successful termination of the reinforcement learner. It communicates back to the metareasoner two things: (a) the policy that successfully solved the problem and (b) the goal for which the policy solved the problem, that is, the policy from discrepancy point to the final goal or to one of the alternative goals extracted from the method preconditions. The metareasoner couples the policy with the description of the discrepancy point, attaches the sensor trace as the expectation for this variant, and constructs a new HTN plan where this policy is succeeded by the partial previous plan starting at the goal method returned by the metareasoner. This algorithm is described visually in Fig. 2.10.

2.3.4 Second experiment

We have implemented our architecture on two different agents in different simulation engines with different sets of primitive actions. The HTN planner plans for a task goal given the primitive actions and assumes a lower-level controller to actually physically instantiate the motion in the agent. The next few sections outline the agent and task specifications and the challenge variation of the environment.

2.3.4.1 Gold hunting in Minecraft

This experiment follows directly from our motivating example of agent Moon. The original task situation is where the two rooms are connected with an open gate, and the variant task is where open gate is replaced by a glass wall. We used the Malmo[g] project for integrating the Minecraft environment events with reward emissions. The Malmo API is also responsible for actuating the physical effects of primitive actions. The primitive actions for this domain are as follows:

[g]https://www.microsoft.com/en-us/research/project/project-malmo/.

1. move forward/backward
2. turn left/right
3. look up/down by 45 degrees
4. break the block in focus

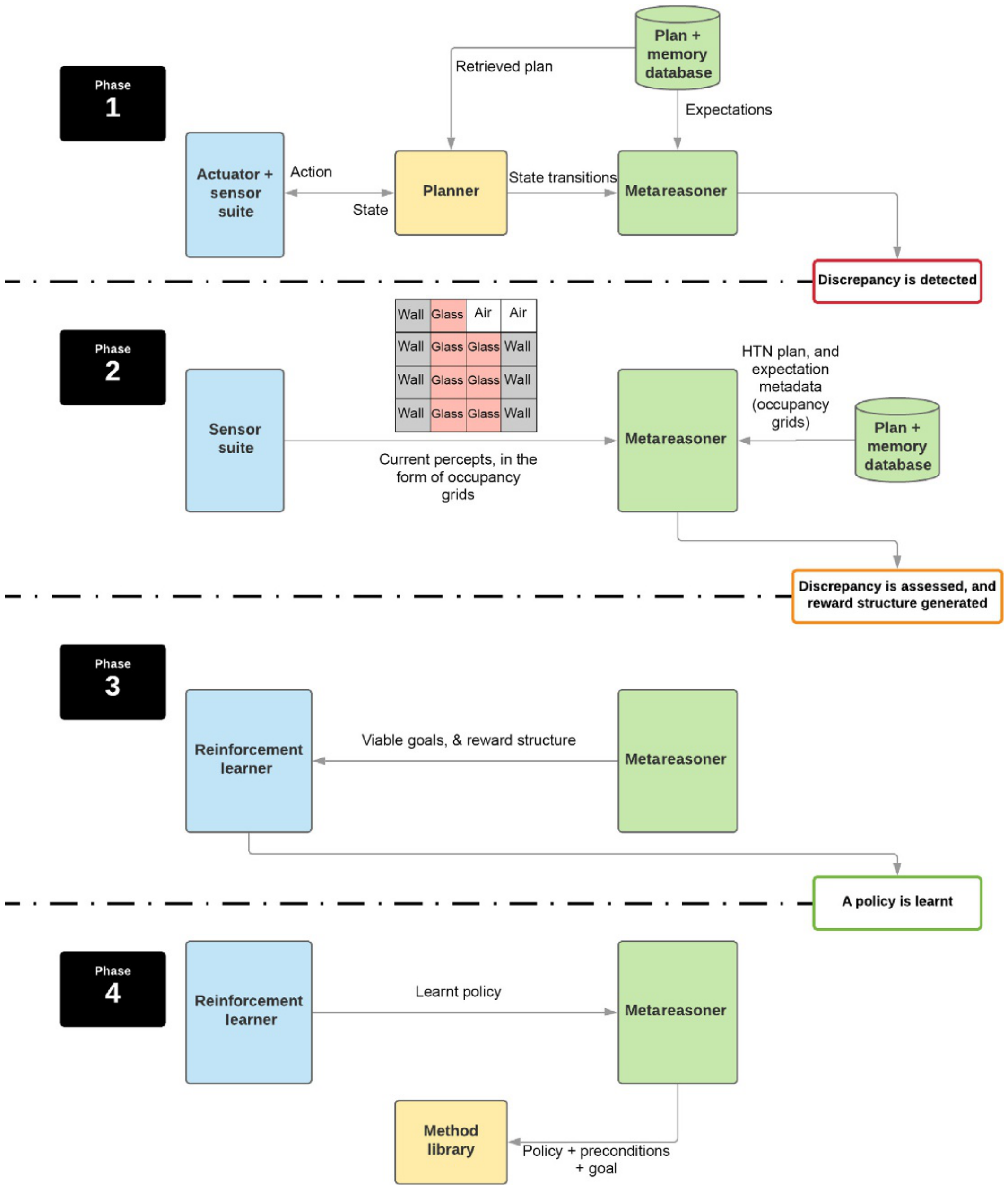

FIG. 2.10 Process diagram: an outline of the communication between different processes.

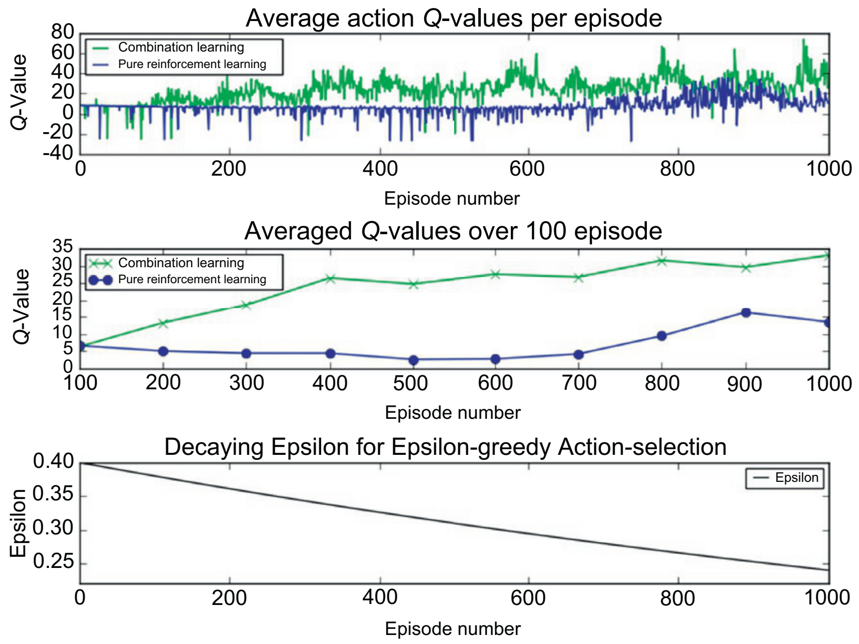

FIG. 2.11 Quantitative comparison of Q-values for the pure reinforcement learning (RL) agent and the combination agent over multiple episodes in Minecraft experiment.

The input state for the reinforcement learner consists of a 3×3 occupancy grid of blocks right in front of the agent along with an array including ground blocks, the block in focus, the in-hand items, and the agent's orientation. The terminating conditions of the learner were either the achievement of any of the goals queued by the metareasoner resulting in positive reward or breaking of any of the walls or ground blocks resulting in a final negative reward. Fig. 2.11 shows the comparison of pure RL against the combination agent. The pure RL agent's Q-values take a dip before gaining value due to a random initial exploration that results in a number of wrong moves. The pure learner presented here is the vanilla tabular learner with the added list of goals queued by the metareasoner, while the combination learner uses the alternate goal list and the random rewards related with making the novel variant more like the original one.

2.3.4.2 *Cup grasping in Gazebo*

The agent in this experiment is a fixed arm in a Gazebo simulator and is tasked with picking up a can placed on the table in front of it (Fig. 2.12). The geometry and relative three-dimensional placements of the object, can, and the table are all known. The primitive actions being used are as follows:

1. Pick object O_i.
2. Place object in hand at (x,y) with respect to the table coordinates.

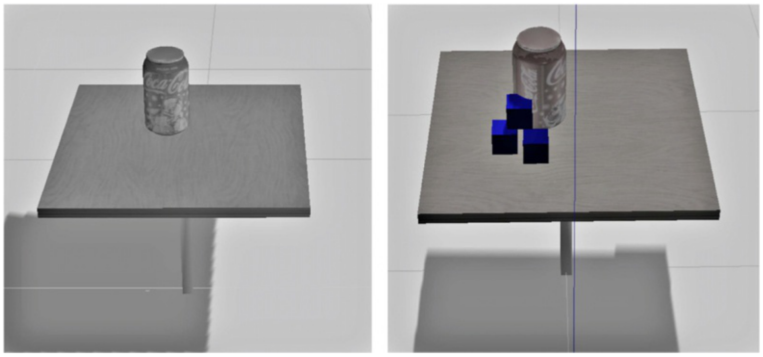

FIG. 2.12 Original *(left)* and variant *(right)* task in Gazebo.

One can imagine the method description for such a domain to be a list of strictly ordered waypoints that were demonstrated or encoded by a human programmer the first time constituent subtasks were executed. In the original version the workspace does not have any other artifacts than the can, arm, and the table. In the variant modification the path of arm end effector toward the can is littered by a random placement of cubes. Since the arm is an immobile and fixed agent, the only way to get around the clutter is to interact with it using our metareasoner. Additionally, just by looking at the primitive actions, it is clear that they are mutually exclusive and dependent on the choice of object and location. Therefore the learning complexity here is grounded in understanding which object to interact with and how. We mention this because usually a Q-learner learns the Q-values with respect to the actions possible, but in this experiment, we learn Q-values in an object-centric fashion, that is, which object has the greater utility in the current situation. The placement location on the other hand is modeled as a Gaussian mixture model (GMM) of those table placements that emitted a positive reward. The learning in this respect is divided between exploring new placements on the table or sampling a placement location from the learned GMM. We notice that, while our algorithm affects the learning for the choice of the object to be manipulated, it does not show any significant improvements to the GMM learning for placement of objects. This result is because the GMM modeling of spatially viable choices already encodes certain knowledge about continuous spaces in it, unlike the chain-like formulation of tabular Q-learning. This result hints at our approach being more useful for domains where the structure and relationships of the state variables are not as well modeled as 2D or 3D spaces.

Fig. 2.13 shows the comparison between the pure and combination learner in the grasping domain. It should be noted that since the number of state variables greatly affect the learning curve of tabular Q-learning, we have supplied the same reduced state (based on configuration space) to the pure learner as we did to the combination learner. This problem is a general limitation of our work as well; matrix computations are expensive, and by storing geometric representations of each past state, certain computations in our algorithm turn out to be computational bottlenecks. However, by combining the configuration space prior with discretized geometry, we limit the upper limits of such expense significantly.

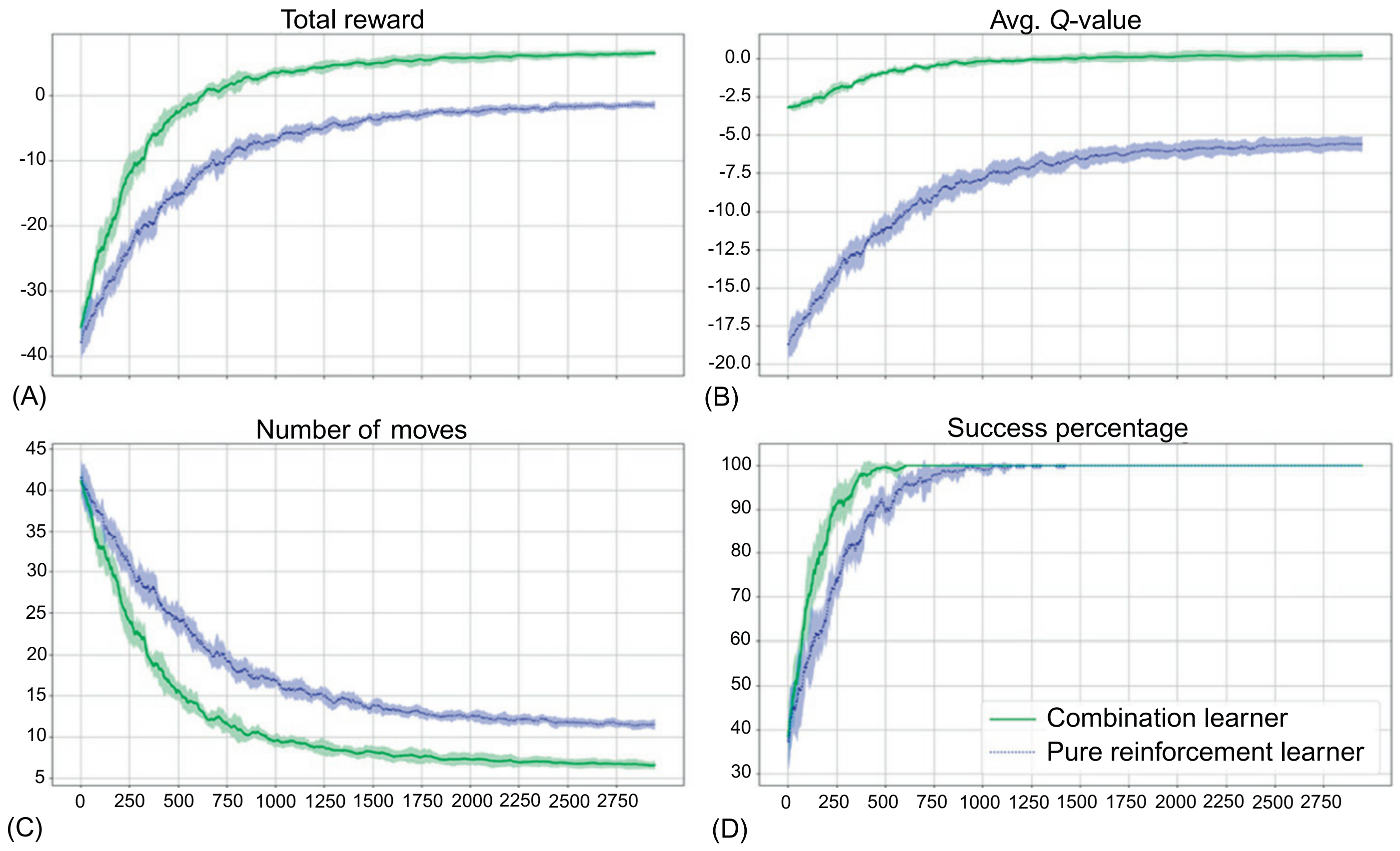

FIG. 2.13 Comparisons over (A) total reward, (B) average *Q*-value, (C) number of moves, and (D) success percentage over multiple episodes for the pure RL agent and combination learner agent in Gazebo experiment.

2.3.5 Summary

This work combines HTN planning and *Q*-learning within the metareasoning architecture. Our results suggest that the metareasoner can use the *Q*-learner 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 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 also potentially for embodied agents that work on real raw data.

2.4 Conclusions

We began this conversation by describing four cognitive strategies for dealing with novelty: situated learning, interactive learning, analogical reasoning, and metareasoning. We said that context determines an agent's choice among these strategies and further that, depending on the context, an agent may also combine these strategies. We proposed that these cognitive strategies may provide a useful starting point for robot design. In particular, in this chapter, we described two cognitively inspired mechanisms for addressing novelty in some detail. In the first mechanism a teacher first demonstrates a skill to an interactive robot, and then, given a new but similar task, the robot addresses the new task by analogy to the known task. In the second mechanism a robot uses metareasoning to combine deliberative planning and situated learning: it first uses deliberative planning to achieve its goal in a new environment and, if it fails, uses reinforcement learning locally to recover from a failure.

What lessons do these robot designs offer for research on cognitive science? The most important lessons pertain to grounding the cognitive strategies for addressing novelty in action and perception. Consider, for example, analogical reasoning. Most cognitive theories of analogy assume representations of knowledge of the world and then run inference engines on the knowledge representations. Further, they assume that only knowledge of some relationships, such as relationships among objects or between goals and methods, needs to be transferred. Our work on the use of analogy in interactive robot learning indicates that there is also a need for transferring the action model from the source case to the target problem, which raises a host of new issues such as the relationships between objects, tools, and tasks; it is just not enough to transfer relationships among objects. Similarly, our work suggests how the knowledge of the world is acquired from perception so that analogies can be made for dealing with novelty.

Similarly, in the case of metareasoning, again, most cognitive theories of metacognition assume representations exist of knowledge of the world and of the agent itself. But our work shows the need for grounding metareasoning in action and perception. For example, the metareasoner may encode task expectations imagistically so that, when it executes a task, it can easily detect failure by comparing the expectation with its actual percepts. Further, it may compile the results of situated learning into deliberative planning for potential reuse. In addition, many cognitive theories assume a specific, relatively static, and rigid architecture for metareasoning. In contrast, our work suggests that a more flexible architecture for situated learning, deliberative planning, and metareasoning works together.

Acknowledgments

We thank William Lawless, Ranjeev Mittu, and Donald Sofge for editing this volume and inviting us to write this chapter; we especially thank William for his encouragement, patience, and support for writing the chapter. Parts of this chapter have appeared earlier in Fitzgerald et al. (2018) and Parashar et al. (2018).

References

Akgun, B., et al. (2012). Keyframe-based learning from demonstration. *International Journal of Social Robotics*, *4*(4), 343–355.

Argall, B. D., et al. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, *57*(5), 469–483.

Bullard, K., et al. (2016). Grounding action parameters from demonstration. In *2016 25th IEEE international symposium on robot and human interactive communication (RO-MAN):* IEEE.

Chernova, S., & Thomaz, A. L. (2014). Robot learning from human teachers. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *8*(3), 1–121.

Chernova, S., & Veloso, M. (2007). Confidence-based policy learning from demonstration using gaussian mixture models. In *Proceedings of the 6th international joint conference on autonomous agents and multiagent systems:* ACM.

Chernova, S., & Veloso, M. (2009). Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, *34*, 1–25.

Dannenhauer, D., & Muñoz-Avila, H. (2015). Goal-driven autonomy with semantically-annotated hierarchical cases. In E. Hüllermeier & M. Minor (Eds.), *Vol. 9343. Case-based reasoning research and development* (pp. 88–103). Cham: Springer International Publishing.

Davies, J., Goel, A. K., & Yaner, P. W. (2008). Proteus: Visuospatial analogy in problem-solving. *Knowledge-Based Systems*, *21*(7), 636–654.

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

Falkenhainer, B., Forbus, K. D., & Gentner, D. (1989). The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, *41*(1), 1–63.

Fitzgerald, T., Bullard, K., Thomaz, A., & Goel, A. K. (2016). Situated mapping for transfer learning. In *Fourth annual conference on advances in cognitive systems*.

Fitzgerald, T., Goel, A., & Thomaz, A. (2018). Human-guided object mapping for task transfer. *ACM Transactions on Human-Robot Interaction (THRI)*, *7*(2), 1–24.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, *7*(2), 155–170.

Gentner, D., & Markman, A. B. (2006). Defining structural similarity. *Journal of Cognitive Science*, *6*(1), 1–20.

Gick, M. L., & Holyoak, K. J. (1983). Schema induction and analogical transfer. *Cognitive Psychology*, *15*(1), 1–38.

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 Intelligent Systems*, *32*(3), 60–67.

Holyoak, K. J., & Thagard, P. (1989). Analogical mapping by constraint satisfaction. *Cognitive Science*, *13*(3), 295–355.

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

Kulick, J., et al. (2013). Active learning for teaching a robot grounded relational symbols. In: *Twenty-third international joint conference on artificial intelligence*.

Lozano-Perez, T. (1983). Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, *C-32*(2), 108–120.

Misra, D. K., et al. (2016). Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research*, *35*(1–3), 281–300.

Murdock, J. W., & Goel, A. K. (2011). *Self-improvement through self-understanding: Model-based reflection for agent self-adaptation.* Amazon.

Nau, D. S., Cao, Y., Lotem, A., & Munoz-Avila, H. (1999). SHOP: Simple hierarchical ordered planner. In *Vol. 2. Proceedings of the 16th international joint conference on artificial intelligence* (pp. 968–973). San Francisco, CA: Morgan Kaufmann Publishers Inc.

Parashar, P., Goel, A. K., Sheneman, B., & Christensen, H. (2018). Towards life-long adaptive agents: Using metareasoning for combining knowledge-based planning with situated learning. *Knowledge Engineering Review*, *33*, e24.

Stroulia, E., & Goel, A. K. (1995). Functional representation and reasoning for reflective systems. *Applied Artificial Intelligence*, *9*(1), 101–124.

Stroulia, E., & Goel, A. K. (1999). Evaluating PSMs in evolutionary design: The autognostic experiments. *International Journal of Human-Computer Studies*, *51*(4), 825–847.

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

Taylor, M. E., Stone, P., & Liu, Y. (2007). Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, *8*, 2125–2167.

Tenorth, M., Nyga, D., & Beetz, M. (2010). Understanding and executing instructions for everyday manipulation tasks from the world wide web. In *2010 IEEE international conference on robotics and automation:* IEEE.

Ulam, P., Goel, A. K., Jones, J., & Murdock, W. (2005). Using model-based reflection to guide reinforcement learning. In *Vol. 107. Reasoning, representation, and learning in computer games*.

Ulam, P., Jones, J., & Goel, A. K. (2008). Combining model-based meta-reasoning and reinforcement learning for adapting game-playing agents. In *AIIDE*.

Waibel, M., et al. (2011). Roboearth-a world wide web for robots. *IEEE Robotics and Automation Magazine*, *18*(2), 69–82. Special issue towards a WWW for Robots.

Further reading

Grounds, M., & Kudenko, D. (2008). Combining reinforcement learning with symbolic planning. In *Adaptive agents and multi-agent systems III. Adaptation and multi-agent learning* (pp. 75–86). Berlin Heidelberg: Springer.

Grzes, M., & Kudenko, D. (2008). Plan-based reward shaping for reinforcement learning. In *Vol. 2. 2008 4th international IEEE conference intelligent systems*, (pp. 10-22–10-29).

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* (pp. 228–241). Berlin Heidelberg: Springer.