6-2009

Intentional learning agent architecture

Budhitama SUBAGDJA
Singapore Management University, budhitamas@smu.edu.sg

Liz SONENBERG

Iyad RAHWAN

Citation
1

Intentional learning agent architecture

Budhitama Subagdja · Liz Sonenberg · Iyad Rahwan

Abstract   Dealing with changing situations is a major issue in building agent systems. When the time is limited, knowledge is unreliable, and resources are scarce, the issue becomes more challenging. The BDI (Belief-Desire-Intention) agent architecture provides a model for building agents that addresses that issue. The model can be used to build intentional agents that are able to reason based on explicit mental attitudes, while behaving reactively in changing circumstances. However, despite the reactive and deliberative features, a classical BDI agent is not capable of learning. Plans as *recipes* that guide the activities of the agent are assumed to be static. In this paper, an architecture for an intentional learning agent is presented. The architecture is an extension of the BDI architecture in which the learning process is explicitly described as plans. Learning plans are meta-level plans which allow the agent to introspectively monitor its mental states and update other plans at run time. In order to acquire the intricate structure of a plan, a process pattern called manipulative abduction is encoded as a learning plan. This work advances the state of the art by combining the strengths of learning and BDI agent frameworks in a rich language for describing deliberation processes and reactive execution. It enables domain experts to specify learning processes and strategies explicitly, while allowing the agent to benefit from procedural domain knowledge expressed in plans.

B. Subagdja (✉)
Intelligent Systems Centre, Nanyang Technological University, Singapore, Singapore
e-mail: budhitama@ntu.edu.sg

L. Sonenberg
Department of Information Systems, University of Melbourne, Melbourne, Australia
e-mail: l.sonenberg@unimelb.edu.au

I. Rahwan
Faculty of Informatics, The British University in Dubai, Knowledge Village, Dubai
e-mail: irahwan@acm.org

I. Rahwan
School of Informatics (Fellow), University of Edinburgh, Edinburgh, UK

1 Introduction

A big challenge in developing a computational system is to make it capable of dealing with changing circumstances. Beyond traditional views of system development, the role of the environment in which the system interacts becomes significant. There is a growing trend to view computer systems as autonomous agents: systems situated in some environment, capable of autonomous action in order to meet their design objectives. An agent needs to be reactive so that it can respond and adapt in a timely manner to changes in the environment. However, the agent should also be deliberative in order to perform tasks appropriately according to its design objectives.

Balancing reactive and deliberative processes is a major issue in building agent systems that deal with a complex changing environment. The complication is that the deliberative processes are computational processes that might take up resources which can reduce the effectiveness of reactivity.

A common framework for building bounded-rational agents, i.e. agents that function appropriately even when computational resources are scarce [15,37], is the BDI (Beliefs, Desires, Intentions) agent model [32]. The model is based on philosophical concepts of practical reasoning in which the reasoning of the agent is directed toward activities like making decisions for performing actions [42]. The model has been successfully applied in application domains that deal with complex dynamic environments [14,17,32]. The model provides intuitive constructs to describe bounded rational behaviors. Based on *folk psychology*,[1] the internal states and knowledge of the agent are represented as explicit mental attitudes such as beliefs, desires, and intentions.

To build an agent, the designer must provide a set of plans as recipes for actions and some initial internal states for the agent in terms of representations of beliefs and goals. In a particular type of model (e.g. PRS [21]), the designer is able to describe the higher level process of deliberation using the same plan representation. Based on this prescribed information, the core agent machinery will go through cycles of deliberation and execution. Based on beliefs, goals, and a set of plans, the deliberation process selects and forms intentions as commitments to produce actions.

The use of prescribed knowledge for actions is the key aspect contributing to bounded rationality of a BDI agent. Plans as recipes can be seen as heuristics that limit the search for appropriate actions. When plans are adopted as intentions, they constrain further considerations of alternatives and thus reduce the need for extensive computational power in the decision making process (e.g. for performing search over all possible plans). In harnessing the limited resources to respond to changing situations, the designer can exploit the structure of information in the environment. Thus, building a bounded rational agent involves analyzing the characteristics of the environment and finding an appropriate set of heuristics.

However, it is always possible that unforeseen conditions will require some modification of the prescribed plans or knowledge so that the performance can be improved and sustained at runtime. Adaptations of the plan repository are still limited to alternating heuristics in a

---

[1] The term folk psychology refers to principles that most people, including laypersons, use to explain their own and other people's behavior in everyday life [25].

reactive manner to find the appropriate one that suits the situation at hand. Most existing BDI agent implementations (e.g. PRS [21], JACK [18], JAM [19], Jadex [8]) assume that plans are unchangeable at runtime. Some platforms, like Jason [4], allow some changes in the plan repository but only in the context of agent communication. The plan is directly received by the recipient agent and stored directly to its plan repository. Another type of plan changes substitute plans as parts of the deliberation and the execution processes such as the one in 3APL [10]. However, none considers the plan changes as an improvement of the agent's performance and learning from experiences at runtime. One reason to keep away from runtime plan modification is to avoid inefficiencies produced by the plan generation processes that may lag behind the changes in the environment, which may render such plans obsolete. Although there are approaches to generating plans using techniques in classical Artificial Intelligence, they are inadequate to deal with interactions and changes experienced in the environment [11,43]. These approaches usually assume the world is static and the only possible changes are only the prescribed results of the actions. These conditions might also contribute to the lack of attention in the capacity for resource-bounded agents to learn.

To make an agent learn from its own experiences without sacrificing many computational resources, one approach is to make learning intentional: to have explicit mental attitudes that motivate and guide the learning. This kind of learning is also known as *intentional learning* [3]. In this case, an extended BDI agent model can be used to realize this kind of learning.

This paper presents a model of an intentional learning agent based on an extension of the BDI agent model. The approach enhances the BDI model with on-the-run plan modification capabilities. The agent is given a collection of plans both for performing in the environment and for learning to improve its performance. The agent is also provided with learning goals that initiate the learning. The incorporation of learning capability in the BDI agent is made to conform with the principle of the use of prescribed knowledge. The agent might make some decisions not just about what to do but also what to learn. In particular, this paper defines the notion of intentional learning by presenting an extension of the BDI agent architecture that makes use of meta-level plans (i) to control deliberation, execution, and commitment entrenchment explicitly; and (ii) to drive learning processes through cycles of abduction or hypothesis testing.

This work contributes to the state of the art in agent architectures in two major ways: firstly to bridge the gap between the BDI agent model and the need for a learning capability; secondly, it provides a new perspective on how to build an agent that is capable of learning. This work combines the strengths of learning and BDI agent frameworks in a rich language for describing deliberation processes. The approach enables domain experts to specify learning processes and strategies explicitly, while still benefiting from procedural domain knowledge expressed in plan recipes.

This paper is structured as follows: Sect. 2 introduces the concept of intentional learning. It provides an overview of how intentional learning can be realized using BDI agent architecture. It also explains the role of meta-level plans as controllers for deliberation, commitments, and learning. The functionality of meta-level plans is extended to control the process of the so called manipulative abduction. Section 3 specifies the structure of the associated mental attitudes and representations to describe learning plans as the building blocks for intentional learning. It also explains how the learning plans can be realized as a kind of meta-level plans and how to design the learning plans. Section 4 presents the empirical investigations on the intentional learning model. It provides some examples of learning plans that have been implemented. It also provides the overview of the results and analysis of the experiments. Section 5 provides discussions of the model in the context of developing

situated bounded-rational agents. It also presents some lessons learnt from the empirical investigations. Section 6 summarizes and concludes this paper.

2 Intentional learning: conceptual framework

2.1 BDI agent architecture

The BDI (Beliefs, Desires, Intentions) agent model [32] is a design framework commonly used in developing agents that behave both deliberatively and reactively in a complex changing environment. The main principle is to use explicit representations of the agents' own mental attitudes (in terms of attributes such as *beliefs*, *desires*, and *intentions*) to direct their actions and selection of appropriate predefined plans.

Beliefs or *belief base* is a data structure that corresponds to a model or knowledge about the world and/or about the agent itself which can be updated directly by events captured by sensors or by some changes in the agent's internal state. The agent's *desires* or *goals* correspond to conditions the agent wants to fulfill. As a *potential influencer* [6], goals may invoke deliberations and/or means-ends reasoning. The agent also has a knowledge structure that specifies how to achieve certain desires or goals in terms of encapsulated procedural descriptions which are stored in the *plans* or the *plan library*. The selected goals and plans to be committed for execution are the *intentions* or *intention structure*. The committed intentions provide constraints on further deliberations and the process of finding ways to achieve goals. Intentions characterize actions and commitment towards future states and guide or constrain future activities including further deliberations or practical reasoning [5].

To deal with a changing world while the agent has insufficient knowledge, plans to be adopted are required to be partial [5,7]. As the world may change, an overly detailed plan about the far future will not be so useful when change in the environment makes the situation no longer relevant with the context of the plan. An adopted partial plan must be filled in with subplans (or subgoals) posing objectives for further deliberation.

As attitudes that characterize actions and further practical reasoning, intentions are specified to have the following properties:

- The intention structure relates executed actions with their motivations (goals) and plans;
- an intention represents the state of execution and/or the progress of the execution;
- intentions have a hierarchical structure posing subgoals and subplans over times.

Basically, the overall control structure of a BDI agent is a cycle which consists of steps like the following:

- Updates beliefs;
- Determines goals at the given circumstances;
- Deliberates and selects intentions to be executed;
- Executes intentions

On each cycle, potential goals are generated that the agent can pursue or achieve. The generated goals are then filtered through the process of deliberation (weighing options and selecting the best one). The selected goal becomes an intention in which the agent commits to execute. After executing the intention, the agent updates its own beliefs based on some external events (e.g. perceptions). Intentions that are already committed can be dropped or abandoned if their executions are successful or believed to be impossible to achieve.

The main challenge in building a BDI agent is to ensure the right portions of time are dedicated to deliberation and the appropriate strategy is used to maintain commitment given

the fact that deliberation is costly. One way to tackle this problem is by arranging the main agent execution (deliberation) loop so that the interpreter can monitor intention execution and decide whether to maintain the commitment to execute the current intention or just drop the commitment and re-deliberate to get a fresh intention. An abstract deliberation cycle adds a particular function to decide whether to reconsider (re-deliberate) the intention at a certain moment of the cycle [23,34,42].

Another approach to dealing with the complexity of balancing deliberations with responsiveness to change is the use of meta-level plans to enable the expression of more flexible deliberations and intention reconsideration strategies. This approach has been used in the PRS agent implementation [13,20,21]. Meta-level plans (Meta-level KAs in PRS's original terms) are plans that operate introspectively on the system's internal state (e.g. beliefs, goals, and intentions) rather than the external world. The meta-level plans in PRS complement the default decision and execution cycle. When certain conditions holds in the agent's internal state, a meta-level plan may be invoked and take over the main interpreter loop for weighing and selecting options. The conditions that invoke meta-level plans might include the applicability of multiple plans (options) in the current situations, the failure to achieve a certain goal, or the reactivation of a suspended intention or goal.

Meta-level plans require access to beliefs about goals and beliefs about intentions. Further, information produced in the deliberation process such as generated options and the selected intention can be exploited to control the deliberation or to accommodate real-time constraints. The kinds of actions contained in a meta-level plan can be characterized as follows:

- *Intention monitoring* A meta-level plan can have actions for monitoring the state of a particular intention at run-time. On some occasions a change in the state of an intention might also trigger a meta-level plan.
- *Controlling deliberations* A meta-level plan can use actions that generate options and select an intention. This kind of actions may also post the selected intention into the intention structure.
- *Managing Commitment* A meta-level plan can have actions that modify or manipulate intentions on the run. These actions can stop an active intention or re-schedule a failed intention if the agent still believes that maintaining the commitment is still worthwhile.

The use of meta-level plans for managing deliberation and commitment is the approach that is used for the agent model in this paper. The next subsection extends the approach of using meta-level plans to conduct learning.

2.2 Meta-level plans for learning

An agent needs to be able to learn from its own experience in order to tackle the problems caused by incomplete or invalidated information. However, most machine learning techniques suggest the application of a particular learning algorithm to optimize the performance of the system [1,28]. This implies that the process of learning and the use of the learnt knowledge are separate parts and the learning process is monolithic. It is a common view in machine learning that experiences are samples to be learnt that are given a priori either by a pre-collection of cases, pre-defined learning scenarios, or some direct supervision from an instructor. Even most reinforcement learning techniques that learn action policies directly from experiences still rely heavily on a preset reward structure during the learning process [39].

In this paper, we use a different approach in which learning is conducted autonomously and intentionally. In psychology, the term *intentional learning* refers to cognitive processes

that have learning as a goal as opposed to learning as an incidental outcome [3]. To learn intentionally one must be aware of one's own knowledge. Bereiter and Scardamalia [3] state that:

> Having a learning goal presupposes that one can prefigure a state of knowledge that one does not currently posses. Thus, one's knowledge about different kinds of knowledge is relevant to intentional learning, determining the kinds of learning goals that one can envision. (p. 373)

The characteristics of *intentional learning* from the psychological point of view imply that learning can be regarded as a kind of behavior that is driven by mental attitudes like goals, beliefs, and intentions. Further, intentional learning requires strategies or know-how about how to accomplish the learning goal effectively.

In cognitive science, a similar view in goal-driven learning [30] has also suggested that the learning process should be guided by reasoning about the information that is needed to serve the learner's goals. Learning is triggered when a reasoner needs to learn to improve its performance in some task. The agent determines *what* to learn based on the information it needs and *how* to learn by weighing alternative learning strategies in the current circumstances.

In the BDI architecture, knowledge for learning can be explicitly represented as a collection of meta-level plans. Using goal representation, it is possible to describe explicitly what conditions trigger the learning process. The conditions for learning can be based on changes in performance based on a certain progress indicator. It is possible to represent many different types of decision procedure when the plans are able to access information about mental attitudes. The expressive power of meta-level plans also enable the description of strategies for intentional learning. To describe what to learn by the agent, the agent designer must provide it with learning plans, specified with the following components:

- *Learning goals* As a meta-level plan, a learning plan can have a goal specified in terms of a condition in the internal state of the agent. An example of a learning goal is the improvement of the performance of an intention based on its progress indicator (e.g. reduction of the number of failures, time reduction of achieving an intention's goal, increasing the intention's success rate etc.).
- *Learning activities* As a meta-level plan, a learning plan can be described to have a body that contains a series of different kinds of actions like intention monitoring, controlling deliberations, and/or managing commitments. Other kinds of operations include plan modification, and belief updates. A more detailed specification of plan creation and modification will be given in the next section.
- *Learning contexts and applicability* Just like any other plan, a learning plan may have preconditions and/or utility. This information is used in the process of deliberation and means-ends reasoning. This also means that a learning plan can be adopted through the process of deliberation and selection.

Just like any other domain level goal, a goal for learning can invoke more than one learning plan but only one plan is selected to be executed. To focus the selection of the applicable learning plan, the precondition attribute can be filled with a condition concerning the status of a particular intention in the intention structure. The structure of intentions, together with intentions' status and progress indicator can be used as clues for determining which hypothesis and learning plans should be selected. These clues can be used as preconditions of plans.

To observe the state of intentions and executions on the run, some primitive actions must be defined and used to support the mechanisms of deliberation and intention monitoring. A

meta-level plan can be made to record decisions from past attempts and use them to avoid making the same mistakes. To realize more complex adaptations, the meta-level plan may incorporate features like plan adaptation and manipulation.

2.3 Learning as manipulative abduction

In this paper, we propose an approach for intentional learning in BDI agents that incorporates abduction into the deliberation process. Abduction is a pattern of reasoning in which hypothetical explanations are formed and accepted. As coined by C. S. Peirce, the process of abduction consists of hypothesis creation and hypothesis evaluation [26]. In particular, there is a type of abduction called *manipulative abduction* which involves experiences and exhibitions of behavior to find some regularities in the environment.

Abduction has been used as one kind of reasoning in artificial intelligence and logic-based systems [22,35]. As opposed to the forward reasoning method, such as *deduction*, which infers the effects from some given causes, abduction produces plausible causes that can explain the effects [35]. In the context of actions, abduction can be considered as planning or means-ends reasoning [36] in which the plan is constructed by inferring a plausible series of events that can lead to the desired states.

Although, our approach uses abduction, there is a difference with the common view of abductive reasoning mentioned above regarding how to apply the abduction. The abduction in our framework is applied throughout a series of actions. These actions may acquire plans as recipes when they are executed. In manipulative abduction, it is the action executions that provide the necessary information given the agent's lack of knowledge. It happens when the agent is thinking *through* doing and not only about doing. Actions in this case are not only for changing the environment to desirable states or satisfying the agent's goals, but they perform epistemic roles. The agent must pursue some actions first in order to acquire the knowledge of how to achieve the goal. This approach in essence parallels with the intentional learning point of view in which learning is considered as a process involving actions to accomplish a certain learning goal.

Figure 1 illustrates the difference between abduction as reasoning and the manipulative abduction process. It shows how the agent changes its beliefs about how to achieve its goal. The normal abduction process (Fig. 1i) involves the construction of a series of actions that may lead to the goal. This reasoning must be conducted prior to the plan's execution. In the example, when the agent desires to get the goal (illustrated as the heart symbol) and it perceives something (the lightning symbol), the agent creates a hypothesis that to bring about the goal while lightning is perceived, several steps of actions must be taken.

On the other hand, manipulative abduction (Fig. 1ii) uses the actions themselves in order to construct the plan. The agent does not have to create a complete plan at the beginning; rather, the plan can be incrementally built based on the actions taken. The example shows that after each action execution, the agent updates the plan by making a hypothesis about how the plan can be developed. Only when the goal is achieved will the agent conclude that the plan works and can be used as a recipe for the next situation when the agent wants the same goal.

Magnani describes that the process of manipulative abduction exhibits some regularities which also implies that some templates of behavior are applied for driving the processes [27]. The templates are conjectural which enable epistemic actions for testing some hypotheses. Various contingent ways of epistemic actions are allowed to apply so that the agent can observe the situation from different perspectives; checking different information; comparing
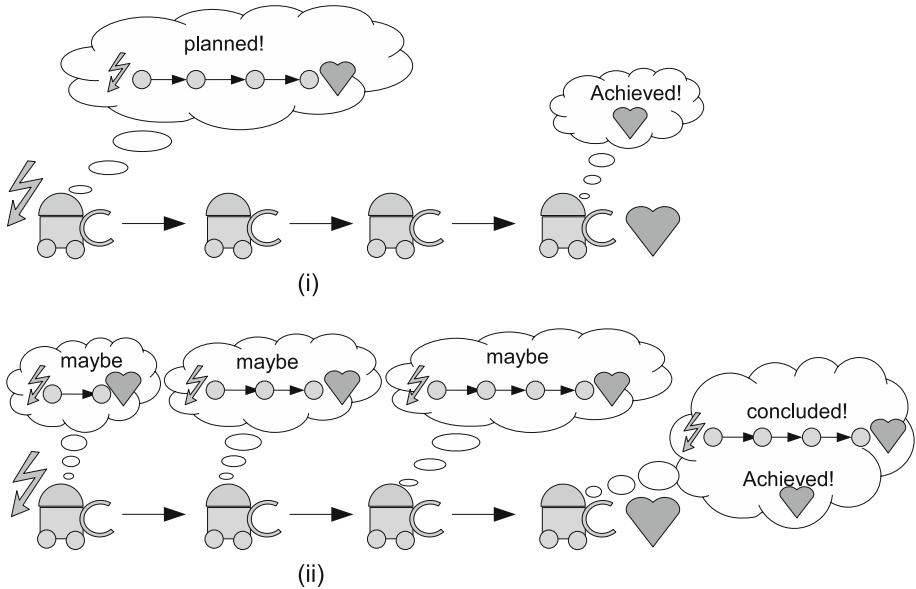
Fig. 1 (**i**) Abduction as a reasoning to construct a plan to achieve a goal state at the start of or prior to executing the plan; (**ii**) Manipulative abduction as a series of actions that construct the plan through executing the actions

subsequent events; and even reordering and changing temporal relationships by evaluating the usefulness of a different sequence. As the templates are conjectural, they are not immediately explanatory. Hence, some templates may require to be selected in the set of available prestored ones, though it is also possible to construct them from scratch.

We suggest that the conjectural templates for manipulative abduction mirror the concept of meta-level plans. In the BDI agent model, some meta-level plans can be described to exhibit the process of manipulative abduction. These plans can be learning plans when they are set to be triggered and enacted when there is a need for better performance or improvement in the future. The hypotheses to be tested and adopted are plans for achieving the domain level goal. These hypothetical plans may be formed and constructed by involving some complex action structures. Based on the hypothesis, a learning plan observes the status and the progress of a particular intention to assess the usefulness of the conjecture. The hypothesis can be discarded when there is still lack of evidence that the hypothesis can be supported (i.e. that it can achieve the given goal). In that case, the BDI interpreter may select a different learning plan which might employ different templates of manipulative abduction.

Using the manipulative abduction process, learning is conducted as part of the agent's activities in the environment. As we shall demonstrate below, however, the characteristics of the domain tasks and the pattern of interactions between the agent and the environment determine how effective a hypothesis and the abduction process can be. This approach of intertwining actions and learning contrasts our intentional learning model with other learning algorithms which are assumed as separate processing from the domain performance. By using this approach, we suggest that given limited time or resources, it is still possible for the agent to adapt by relying on some prescribed recipes. The intention to learn can improve the agent performance and may take effect in a long term rather than just as an instant decision making.

The primary motive of using manipulative abduction process in intentional learning is to enable learning when computational resources are limited so that planning from scratch is not feasible. This kind of learning may actually be a rather risky and time-consuming process when it is applied in an inappropriate context or situation. As a part of the mechanism of a BDI agent, the manipulative abduction process should be started and applied intentionally when the agent believes that starting the learning is currently reasonable and expects that some positive results will be produced by applying the manipulative abduction.

As the first step toward realizing the intentional learning agent architecture as a new model of learning mechanism, we develop the model and show some templates of learning plans as examples of the intentional learning can be made.

3 Intentional learning: recipes for learning

In this section, we show how to realize the intentional learning mechanism represented in BDI agent architecture. We demonstrate that the BDI agent model with explicit plan representation provides more control and flexibility in describing learning process. In particular, through the manipulative abduction, a sequence of meta-level actions can be executed to test a hypothesis while in the meantime low-level actions gather additional information for developing the hypothesis. Learning is conducted as part of the agent activities in the environment using the same mechanism of execution as in ordinary plans. Using a conjectural template or hypothesis, a learning plan can be used to describe a course of actions that affect not just the state of the agent within the environment but also the state of the agent's mind after the actions. Just like an ordinary plan, the execution of a learning plan may be interrupted, suspended, or dropped when certain events occur. Based on the status of improvement or the state of knowledge, the learning may also fail or succeed.

By the manipulative abduction process, the form of knowledge acquired corresponds to the form of hypothesis made in the abduction process. The conjectural template or the hypothesis can be assumed to be determined already by design. It may be prescribed explicitly in the plan or may be implicitly represented by the series of actions that find some evidence to support the hypotheses. Although this kind of learning process may acquire various types of knowledge for different purposes, we focus on the capability of learning plans to produce procedural knowledge or recipes that also use the same plan representation as a BDI agent mainly relies on plans and deliberation process to behave in its environment.

To find some useful recipes by acting in the environment, a complex structure of actions may also need to be described that guides the agent to evidence the hypothesis. By manipulative abduction, the agent does not just attempt to confirm a single hypothetical condition, but may also involve a monitoring process on a complex structure of actions and events. Furthermore, subsequent meta-actions in a learning plan my be interdependent with one another in a way that each action may use or produce information that can be exchanged between different actions. This interdependency is needed especially when the agent constructs a new plan involving an intricate structure of actions. In other words, a learning plan must be designed and constructed to be composed of actions (or meta-level actions) by taking into account how information within the agent's states of mind is affected or exchanged between the actions.

Furthermore, the actions of a learning plan may use or produce information that may be exchanged with different actions inside other plans. This means also that learning plans may be interdependent with each other or with other domain level plans. To design and construct an effective learning plan, it is important to take into account how information about the tasks

and the agent's states of mind are exchanged between actions and how they can affect the actions' execution.

This section describes in more detail how intentional learning with learning plans can be realized. However, it is not just the representation structure that matters. The patterns used to describe the learning process and plans are significant factors that may determine the effectiveness of the approach. Therefore, besides showing an overview of the structure of meta-level plans for learning, we will also discuss important aspects to consider when designing learning plans and some examples of patterns for building them.

3.1 The structure of learning plans

Learning plans can be represented in the same way as a normal domain level plan. The difference is the kind of actions involved and the kind of information accessed and processed during the execution. The common structure of a plan includes the following attributes:

- *Goal* the condition that may invoke and activate the plan and/or the target state the plan attempts to achieve.
- *Precondition* the conditions that must hold before the plan can be activated or adopted.
- *Body* a series of actions to be executed when the plan is adopted.

A plan can also have other attributes depending on the context and the domain the plan or the agent is applied to. The model of the plan in this paper also incorporates *utility* as an attribute.

There are different ways to describe a plan's body. In a simple form, the plan body consists of a sequence of actions. Alternatively, the body may contain more complex relations between actions such as repetition, conditionals, concurrency, and so on. Kinny et al. uses a special diagram called *plan graph* which resembles a state transition diagram used in OO dynamic modelling to describe plans [24]. Some later methodologies suggest the standard UML activity diagram to describe plans and actions [2,9]. We use a *plan schema* to describe plans and adopt the UML activity diagram [12] to describe the structure of actions in the plan body. UML activity diagram are expressive enough to describe the features mentioned in the last section. An activity diagram can capture complex relationships and orders between actions. It can also describe information flow between actions. Both complex control flow and object (information) flow models are necessary for describing learning plans as actions may be interdependent with each other.

Figure 2 presents an example of a plan described with the schema notation. Both goal and precondition are expressed as conditions using square brackets which actually correspond to nested propositions or attribute-value pairs. For example, a single symbol inside the square brackets refers to a proposition which is shown in the precondition in Fig. 2. The precondition [front-empty:] means that if the agent believes that there is no object that obstructs the agent in front of it then the plan is applicable. The goal attribute [achieve: [condition: sensed-object:]], on the other hand, means that the agent wants to achieve a condition that an object is sensed (sensed-object).

The activity diagram is used as the notation for describing actions in the body attribute. A rounded rectangle denotes an action, while a rectangle represents an object or information. A control transition between actions is denoted by a solid arrow line. The dashed arrow line denotes the flow of information or objects. A dashed arrow line outgoing from an action to an object means that the action changes the information of the object. On the other hand, a dashed arrow from an object to an action means the action uses information from the object as input. The control transition can also have branches
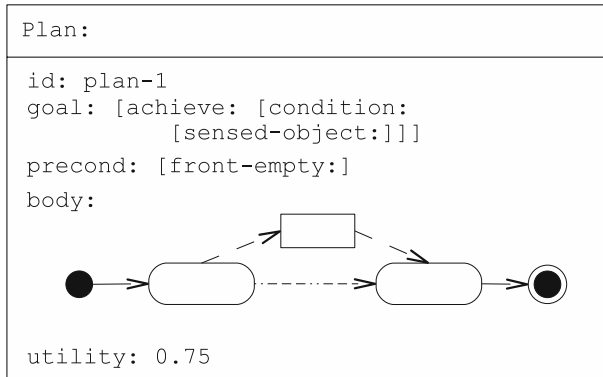
```
Plan:

id: plan-1
goal: [achieve: [condition:
            [sensed-object:]]]
precond: [front-empty:]
body:




utility: 0.75
```

Fig. 2 An example of a plan schema. The plan has attributes like `goal`, `precondition`, `body`, and `utility`

representing choices or conditional transitions using a diamond-shape symbol. A branch can also be used to construct a repetition by putting the direction of a transition back to an earlier action.

In this paper, the syntax of the activity diagram is used only for illustrative purpose. We do not intend to define specifically the semantic of symbols in the diagram as there can be different types of semantic and interpretation for the UML activity diagram and each may be based on the problem domain and how the designer addresses it. In most parts of the paper the semantics of the diagrams presented are rather loose and may be combined with different notations to express some aspects that can not be covered by the diagram alone.
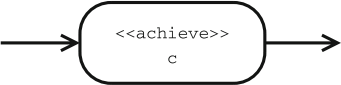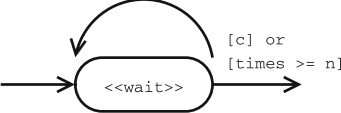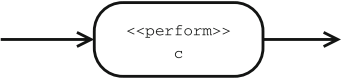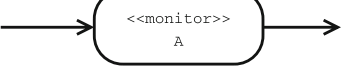
3.2 Meta-level plans

As mentioned earlier meta-level plans require access to beliefs about goals and beliefs about intentions. Further, information produced in the deliberation process such as generated options and the selected intention can be exploited to control the deliberation or to accommodate real-time constraints.

Intention monitoring and commitment management in a meta-level action can be realized directly using the specified primitives and the action structure. For example, to identify the actual conditions of the intention structure, some primitive actions can be used to obtain information about the state of the intention structure based on a complex query expression. Table 1 shows stereotyped (has a more specific meaning from UML standards) actions that are used for monitoring and managing intention and execution.

Each action in Table 1 returns a data structure which consists of the status of the intention or the actions it monitors. For example, both achieve, and perform actions store and update the number of trials in a variable `trials` and they also record all actions (plans) selected in a subsequent order in a variable `sel-history`. The monitor action, on the other hand, does not record the history but stores and updates the status of actions it monitors in a variable `sub-result`. These particular variables or data structures can be passed on to another action to be processed further.

There are also some other primitives that are not stereotyped but important for meta-level plans. The `deliberate` action gets its input from a description of an intention and produces a data structure about the choice it makes. The data structure produced can also be called *meta-level attributes* which can be described as follows:

Table 1 Special primitives for meta-level plans

No.	Diagram notation	Description
1.	`<<achieve>>` `c`	Achieve action is an action that posts a subgoal and triggers the deliberation and plan selection for achieving the condition `c`. After the intention for achieving `c` ends, the condition `c` is checked if it is true. Otherwise the achievement subgoal fails.
2.	`<<wait>>` [c] or [times >= n]	The wait action halts or suspends the condition until the condition `c` holds or optionally the number of cycle in variable times reach a number `n`.
3.	`<<perform>>` `c`	The perform action directly adopts a plan and executes it using the plan identifier specified in `c`.
4.	`<<monitor>>` `A`	The monitor action executes a description of actions and return its status (fail or success) but the monitor action itself is always succeeds.

- `apl`: list of applicable plans.
- `intent`: the current intention. If it is empty at the beginning, its value is created with values taken from the current goal.
- `history`: list of plans that have been selected in previous trials.
- `plan`: the current selected plan.
- `goal`: the current goal. This attribute will be filled with the goal from the intention attribute if it is empty at the beginning. The goal may also be used to fill up the intention attribute if the intention attribute is empty at the start.
- `implement`: action structure to be attached as the subintention as a result of the deliberation process.

The values of meta-level attributes can be created, manipulated, or exchanged between actions in a meta-level plan for controlling deliberations. The `deliberate` action is also a customizable action in which different methods of deliberation and decision making can be used to extend or override the default mechanism. The `subgoal` action is also a kind of action for the deliberation process. It posts a subgoal of a given intention directly and waits until it finishes. Its status of success (or failure) corresponds to the status of the subgoal it monitors.

The body of a meta-level plan can express different kinds of decision making procedure and can be of arbitrary complexity. Because it is represented and executed in the same manner as any other plan, other kinds of actions that are used in domain-level plans can also be applied as parts of the body of a meta-level plans in addition to the kinds of actions mentioned in the specifications above. The adoption of an intention can also be conducted locally within a meta-level plan using the existing actions schema like `perform` or `monitor` actions structure. In that case, the meta-level plan can be made as an independent controller for deliberation and execution that overrides the default execution cycle.

The introspective capability, made possible by the above primitives, is important for the learning process in which adaptation can be made to align with the current needs of the agent so that the learning can be more focused and directed.

3.3 Designing learning plans

Just like a normal intention, a learning intention can be hierarchical as well. A learning plan can post a subintention which is the domain level intention to be monitored and improved. While the subintention is executed during the learning process, some subgoals might be posted and further deliberations and/or trials would be invoked as well. This also implies that another learning plan would be selected and executed at a subordinate level if the performance of the subintention degrades whatsoever. Different learning strategies can be applied on different levels of the intention structure.

A goal for learning can invoke more than one learning plan but only one is selected. The structure of intentions, together with intentions' status and progress indicator can be used as clues for determining which hypothesis and learning plans should be selected. These clues can be used as preconditions of plans. For instance, the learning treatment for a sequence may be different from handling the choice structure. In another case, a different number of times a particular intention is tried may lead to a different hypothesis (e.g. a different strategy should be applied when the same intention has been tried for more than certain times without success). Moreover, different hypotheses can be made for different relationships between intentions in the hierarchy. A failure (or success) of an achieve intention may be caused by a failure in its sub-intention. However, a failure can happen also because the condition specified in the achieve intention still does not hold at the end even though its sub-intention was done successfully. Both cases may require different hypotheses and learning strategies. Primitives for meta-level plans mentioned above can be used to observe the state of intentions and executions on the run.

The plan exhibits an adaptive process in which the decision to select a plan depends on past experiences. The role of recording the decision made from previous attempts is to avoid making the same mistakes. However, to realize more complex adaptations, the meta-level plan needs to incorporate features like plan adaptation and manipulation.

3.3.1 Learning primitives

To enable the agent to adapt its own plans on the run, a set of primitive actions or operators specific for asserting, updating, and deleting plans can be provided for meta-level plans to support the plans adaptation. In this paper the detailed specification for plan manipulation primitives is not given as it may also depend on the implementation issues such as how the plan is actually represented and stored. A list of plan manipulation actions is provided in Table 2 together with a description and definition of each input and output.

Just like meta-level primitives described in Sect. 3.2, a plan manipulation primitive may access or update meta-level attributes. The values of meta-level attributes can be created, manipulated, or exchanged between actions inside a learning plan or across different meta-level plans. These values can be used for constructing and manipulating plan attributes during the execution of a learning plan.

There are different ways to learn a plan using the manipulative abduction. One approach of plan manipulation is to build or manipulate the plan *in-place*, i.e. when the plan is still in the plan repository. The creation and changes are applied directly to the stored plan so that the change may just directly take effect. Another approach is *out-place* which instead creates or changes a plan outside the plan repository and retains the change as a separate data structure that can be passed between actions. The effect would only be occurred at the end

Table 2 The specification of the plan manipulation actions

Type	Action	Input	Output	Description
Plan manipulation	`[create-plan:]`	`[schema:]`	`[plan:]`	Creates a new plan in the plan library
	`[update-plan:]`	`[old:]; [new:]`	`[plan:]`	Update the plan that matched with the old with attributes of plan in the new one
	`[remove-plan:]`	`[old:]`	`[plan:]`	Remove a plan in the plan library that matched with the old
	`[make-plan-schema:]`	`[in-plan:]; [goal:]; [precond:]; [body:]; [util:]`	`[plan:]`	Construct a plan schema based on attributes available (goal, precond, body, and util)
	`[append-plan-acts:]`	`[in-plan:]; [acts:]; [cond:]`	`[plan:]`	Append an action or actions structure at the end of the actions list of the body of the plan
	`[insert-plan-acts:]`	`[in-plan:]; [acts:]; [cond:]; [pos:]; [order:]`	`[plan:]`	Insert an action or actions structure at a specified position in the body of the plan
	`[remove-plan-acts:]`	`[in-plan:]; [pos:];`	`[plan:]`	Remove an action or actions structure at a specified position in the body of the plan
	`[inc-plan-acts-times:]`	`[in-plan:]; [inc-fact:]`	`[plan:]`	Increment the number of times of a repetition structure in the body of the plan
	`[dec-plan-acts-times:]`	`[in-plan:]; [dec-fact:]`	`[plan:]`	Decrement the number of times of a repetition structure in the body of the plan
	`[set-plan-acts-times:]`	`[in-plan:]; [times:]`	`[plan:]`	Set the number of times of a repetition structure in the body of the plan
	`[set-plan-rep-cond:]`	`[in-plan:]; [cond:]`	`[plan:]`	Set the condition of a repetition structure in the body of the plan
	`[set-plan-acts-else:]`	`[in-plan:]; [acts:]`	`[plan:]`	Set the actions of the else part of a conditionals structure in the body of the plan
	`[inc-plan-util:]`	`[in-plan:]; [inc-fact:]`	`[plan:]`	Increment the utility value of a plan
	`[dec-plan-util:]`	`[in-plan:]; [dec-fact:]`	`[plan:]`	Decrement the utility value of a plan
	`[get-action-plan:]`	`[in-plan:]; [index:]`	`[act:]`	Get an action step from the actions structure of a plan at a given position

Table 2 continued

Type	Action	Input	Output	Description
	[app-merge-acts-plan:]	[in-plan:]; [acts:]	[plan:]	Merge actions of a structure into the end of another structure in the body of the plan
	[ins-merge-acts-plan:]	[in-plan:]; [acts:]; [pos:]; [order:]	[plan:]	Merge actions of a structure starting at a given position in the body of the plan
	[split-acts-plan:]	[in-plan:]; [pos:]; [order:]	[left-acts:]; [right-acts:]	Split the actions in the the body of the plan into two parts

of the learning process when a learning primitive explicitly stored the plan data structure to the plan repository.

In-place manipulation can be used to make the learning takes effect even though the learning process fails. In this way, the learning is more memory efficient as less information is required to be passed on between actions. However, the approach may also be unstable to the system as a false plan may be learnt and the source of error may be intractable. On the other hand, the *out-place* adaptation can be a safer choice as the effective change will only occur when the learning plan is explicitly invoke it. However, the performance of learning to get the right knowledge would take longer as the learning plan may fail many times and no improvement can be made to the plan repository before it succeeds. This characteristic of the *out-place* adaptation is also indicated in one of our the empirical investigations (Sect. 4.1). The *out-place* adaptation may also take up more memory space as the learning process needs to retain the information of the plan currently constructed or manipulated outside the plan repository. Both kinds of plan manipulations are used in our experiments which will be described in the later section.

3.3.2 Types of learning plans

The main feature of the approach of intentional learning in BDI agents is that it produces recipes for actions. As recipes, the acquired knowledge compactly represents guidance for performing the wanted actions or achieving the goal conditions rather than a complete model of the world. By building recipes for actions, the learning also conforms to the resource bounded principles of the BDI agent in which the agent learns to act in a focused but reactive manner.

A learning plan can be characterized in terms of the form of knowledge (or plans) that can be acquired. The form of knowledge acquired corresponds to the form of hypothesis made in the abduction process. Another kind of characterization can be based on the method of hypothesis confirmation. The hypothesis about the plan structure can be based on the characteristics of interactions between the agent and the environment or the task domain. As guidelines to create a learning plan, there can be different characteristics of the environment that the designer of the learning plan can assume to hold. Those characteristics that can be the basis for building a learning plan are as follows:

1. The task environment is fully perceivable. In this environment, the agent can perceive all necessary information that distinguish one state from another.
2. There are hidden states in the task environment. In this situation, there are some states that can not be perceived or distinguished from one another by the agent because of limited perceptions or sensing capabilities.
3. There is an ordered or sequential dependency between states. In this situation, there are states that can only be accessed if the agent has reached some other states.
4. The environment is changing independently. In this situation, there is an object or another agent that actively changes the state of the environment.

This list of characteristics can be considered as a subset of a more generalized properties of task environments for agents [33]. Some less relevant properties are omitted such as discreteness and multi-agency. However, it is still possible to include those dimensions in the future when learning about them can be beneficial for improving performance. A learning plan can be built on the basis of one or more of the conditions listed above. Each characteristic can determine the kind of plan produced from the learning process. The first condition

is suitable when the plan produced by the learning resembles a reactive rule in which a certain condition triggers the application of a single action. This is because the agent can rely solely on its sensory or perceptual capability for making a decision for the appropriate action. After executing the action, the agent can then make the appropriate decision again by relying on the perception. Using characteristics 2 and 3 requires the learning to produce plans with a complex structure of actions. For instance, item 2 is suitable for a plan with repetitive actions. The learning plan can use the assumption that the state is changed after executing an action although the agent can not perceive the change. Doing the same action repetitively can take the agent to another state through some trajectory. In another case, using the assumption in item 3, a plan with a sequence structure may be more suitable. Before getting into a certain state with the action A, the agent must do something else to reach another state so that the action A is applicable. In addressing the situation described in the last item (item 4), the learning plan can be made so that the agent finds a local solution like preparing an escape or avoidance plan as an anticipation at a certain point in a plan when the change that brings the inconsistency happens. A strategy exploiting subgoals can be useful in this case.

Based on these conditions, a learning plan can be characterized based on what kind of knowledge or plans it creates or manipulates. Two kinds of learning plan can be characterized as follows:

- *Reactive rules acquisition* One simple form of knowledge to be acquired is the one that can exhibit reactive behavior which always determines the current action based on the current perception. A plan structure can accommodate this reactive behavior by filling the precondition attribute with a condition of a perceptual input, and the plan body consists of a single step action.

- *Complex actions acquisition* The learning plan projects a sequence of actions or a complex structure that takes up some set of the execution cycles instead of only a single action. The meta-level plan representation allows the expression of learning patterns that can make plans beyond simple reactive rule-like structure or plans with single action steps. As a plan for manipulative abduction, a learning plan can also select an incomplete or partial hypothesis which may not be immediately testable. The hypothesis may be developed or built up during the plan execution through some complex structure of actions (e.g. sequences, iterations, subgoals, choices and so on). This approach makes learning a complex action structure possible.

The reactive rules acquisition can be made to align with reinforcement learning which is the common mechanism for agents learning. Each decision to learn a reactive plan can be based on the possible rewards the agent would get. The learning plan for reactive rules may also be able to adjust the utility of a plan being learnt. A successful application of a plan can increase the plan's utility and the adjusted utility value may be propagated to the value of another plan that has been applied just before the successful one. It simulates the reinforcement learning in which a rule or a state that receives the reward will increase the value of another rule or state that is believed to have contributed the attainment of the reward. In this case, the rewarding state can be considered as the goal state.

This does not mean that this form of intentional learning is equivalent with reinforcement learning. In fact, the intentional learning goes beyond what is already possible with reinforcement learning by providing a more flexible and expressive notation for describing learning. In this paper, the reinforcement learning is simulated as a learning plan for demonstration and evaluation purposes only.

Furthermore, using reactive plans may instead slow down the execution, as a single step of action will always be followed by a deliberation. In contrast, the original motivation of the BDI agent architecture is to exploit the intention structure and explicit plan description to constrain reasoning for deliberation. In that case, acquiring complex actions structure can be more suitable for an intentional learning agent to behave conforming the BDI model.

3.4 Comparison with other BDI platforms

The ability to control and manipulate plans on the run is not unique to intentional learning. Some BDI agent platforms have used it to enhance the flexibility and the performance of the agent. One example that has exploited meta-actions to control the intention and execution of the agent is Jason [4]. Jason is derived from a formal agent language called AgentSpeak [31], but comprises non-standard meta-actions that allow a plan to monitor the state of current intentions and desires at runtime. The intentions and desires can be changed or dropped using special internal actions. In Jason, it is possible to create a plan to realize the deliberation and commitment strategies. Jason also includes actions to modify the plan library on the run. However, the capability of changing plans is still limited only to adding and removing plans from the plan library, but not comprising the modification of a certain plan structure or capturing the state of the execution at runtime, such as parts of the plan or intention that fail. Similar features have also been realized in some implemented agent interpreters derived from PRS [21], such as JAM [19] and SPARK [29], but they have limitations with similar to those of Jason.

3APL [10] is another intentional agent architecture that uses explicit rules to describe how the plan can be broken down or revised when certain conditions block the current intended plan or the plan selected is incomplete or partial. A plan revision rule can react to modify the intended plan at runtime when certain internal conditions meet. The rules are limited only to support the practical reasoning process in forming the plan to achieve the goal in a particular run but not to improve the performance of the plan in a longer term nor to change the contents of the plan library. On the other hand, the intentional learning model exploits all the meta-actions for execution monitoring, controlling, and plan modification so that a new plan structure may also be created based on plausible conditions in the environment, thanks to the manipulative abduction process. The intentional learning agent architecture offers more flexibility in making the agent capable to find the solution to achieve the goal and use that experience to improve the performance of the agent in a longer term.

The next section will describe some implemented learning plans in more detail. The description is associated with discussions of an empirical investigation in which some learning plans are implemented.

4 Intentional learning: some empirical investigations

The model of intentional learning described above has been implemented for experiments to support the hypothesis that designing learning strategies using a plan representation can be beneficial. Two different experiments were conducted. The detailed explanation of the experiments is given in [38].

Both experiments assume that the learning and the domain level performance are blended together in continuous activity. There is no clear separation between learning cycles and domain level activities. The first experiment is used to investigate the influence of aspects like observability of the environment states and the structure of the plans learnt for simple routing tasks. The second one explores properties of plan acquisition based on different learning plans. The aspects investigated provide examples of how a learning plan can be designed.

4.1 Rat's world

The first experiment domain called the rat's world, is a simulation domain inspired by the psychological experiment of operant conditioning. Consider a situation resembling a laboratory experiment of learning behavior. An artificial rat agent is resided on an isolated place where a cheese is placed inside a transparent box. The rat agent has a desire to get the cheese, but the box must be opened first. To open the box the rat agent must press some buttons in a particular order. There are two possible actions for the agent at one moment: pressing the left and the right button. Each button is characterized by the position (left or right) or its color (black or white). The situation is illustrated in Fig. 3.

However, the button pressing action is associated only with the button's position but not with its color (there is no action such as press the black button or white button). The only way of relating another property of the button, such as color, with the action is to express a more complex description of activity like describing recipes using the plan representation.

To test the intentional learning agent, the environment of the rat's world can be programmed and configured so that the goal state can only be reached based on a certain rule or order. In particular, the cheese box will only be opened if the agent has done a certain configuration of actions, possibly involving a complex dependency between some properties of objects and a configuration of actions (e.g. opening the box when the white-colored button is pressed after the black one). This is the rule that is adopted throughout the entire experiment of the rat's world. The rule is presented using the UML statechart diagram in Fig. 4. The diagram also implies that the task of opening the box is continuously performed.

The main challenge for the agent in this case is to find out the right behavior that conforms with the rule of the environment mentioned above. The difficulty of the problem stems from the fact that that rule of the environment has no direct correspondence to the agent's action representation (e.g. instead of pressing black or white buttons, the agent can only press the button on the left or on the right). Another complication arises when the properties of objects

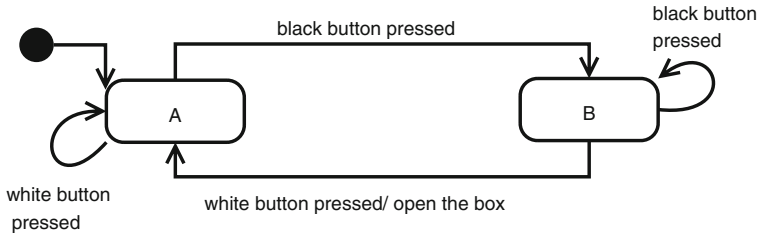**Fig. 3** The rat's world: operant conditioning
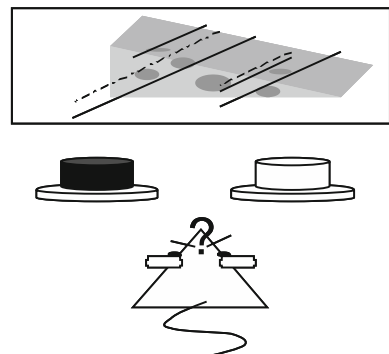
Fig. 4 The rule of the environment in the rat's world in UML statechart diagram

may change over times. In the experiment, another rule is also applied to the environment so that the positions of the buttons may be swapped in a random fashion every time the box is successfully opened. In this case, the agent must be able to deal with dynamic situations as the knowledge that has been learnt from previous experiences may conflict with the condition in a later attempt.

4.1.1 The implementation and learning plans

The rat's world simulation has been implemented based on the integration of two different platforms: The Netlogo multi-agent modeling environment [40] and the JAM BDI agent framework [19]. Netlogo was used as a tool for building the simulation environment. The agent was implemented as a Java class program embedded with the JAM reasoning engine and interfaced with the environment built using the Netlogo. The JAM interpreter resembles the PRS architecture in many ways. It supports the expression of complex action structures and meta-level plans. Unlike other platforms which also adopt some features from PRS (e.g. JACK [18], JADEX [8]), JAM supports a limited capability of re-evaluating plans at runtime. In practice, it can add new plans on the run as a non-standard feature of the language even though runtime modifications or removal of plans are not yet supported.

Both the domain level goal and the learning goal are initially put side by side as parallel goals that are active continuously over time. This can be realized in the JAM language using *maintain* goals for both domain and learning goals. In JAM, a *maintain* goal reattains its goal condition if it becomes unsatisfied or false. This kind of goal will be kept in the goal list even though the goal condition has been achieved. The use of maintain goal also conforms with the assumption of continuous activity in learning mentioned above.

The experiment applies two different main goals: a maintain goal for opening the box expressed in JAM syntax as `MAINTAIN BoxOpen`, and a maintain goal to achieve high reliability in opening the box, which is expressed as `MAINTAIN Reliable`. The agent starts with a belief that the performance is reliable. Over time, when the agent fails to achieve this goal, the performance value is decreased. If the value becomes under a certain minimum threshold, the belief in reliability will be retracted and the learning plan will be activated. The learning process, then, runs in parallel with the domain level actions to monitor the status of the execution and construct new plans accordingly. If the plans constructed produces more successful attempts, the performance value may increase and the reliability belief will be re-asserted, which can stop the activation of the learning plan. This also implies that the type of learning actions in the rat's world is the *out-place* manipulation in which the plan learnt will not take effect until the whole steps of execution succeed and the performance improved.

The performance level of the box opening task is measured by an external process implemented in Netlogo which reports back the performance level to the agent's beliefs as sym-
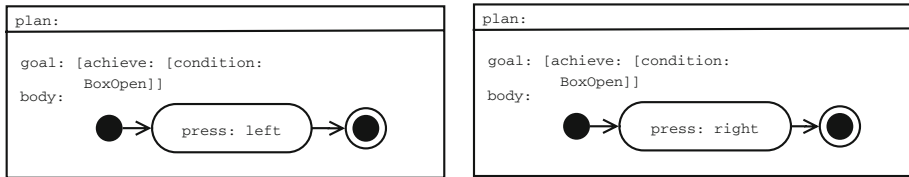
Fig. 5 Default plans for opening the box in the rat's world

bolic statement. There are four symbolic levels for characterizing performance in the agent's beliefs: LOW, MED-LOW, MED-HIGH, and HIGH. In the experiment, the performance value is the average of successful attempts within a period of time which takes a value between 0 and 1. It can be calculated based on the ratio between the number of successful attempts and the number of trials $p = 2N/T$, in which p is the performance value with N and T the number of successful trials and the total number of trials respectively. The N value is doubled before dividing by T so that the performance value will be 1 (maximum) when two different buttons are pressed consecutively in a certain order.

By default, two different initial plans are used to open the box. Figure 5 shows the two default plans in the schema notation that is used for describing the intentional learning model.

The deliberation cycle in JAM selects one of the two plans at random as both can be assumed to have the same utility. In every moment the agent perceives the position and the color of the button. The belief about the buttons can be expressed as [buttons :[left :c_{left}] [right :c_{right}]]. The c_{left} and c_{right} refer to the color of the button on the left and the right respectively. To deal with hidden states that can not be observed by the agent, the agent also has beliefs about the past action and result which is expressed as [last_act: [act: a] [stat: s]]. The action a in the act attribute is the last action taken which can be [press : left] or [press : right]. The attribute stat consists of the result status s of the action a which can be SUCCESS or FAILS.

We used two kinds of learning plans in the experiment. The first is a learning plan for constructing reactive plans, and the other is a learning plan to construct plans with a complex action structure. The reactive type of learning plan can be broken down into two categories: simple condition-action reactive plans and condition-action with intermediate plan. The plan with complex actions structure can also be classified into two: a sequence with simple observational precondition and a sequence with mixed observation-introspective precondition.

The learning plan with simple condition-action is presented in Fig. 6. This plan is a kind of reactive-rule acquisition plan. The plan works by constructing new plans based on failures captured on the run. A new plan is constructed using a perform subplan construct-react-plan to construct and store the new plan.

The plan in Fig. 6 is activated when condition Reliable is not true in the belief base. The learning plan queries the status of the failed action and the buttons (stored in act1 and perc respectively). Then it waits until there is a successful action during the execution of the domain level plan (stored in last_act). As soon as the action succeeds, a new condition-action plan is constructed using the action construct-react-plan, which uses the three variables above.

The action construct-react-plan in the learning plan of Fig. 6 actually activates the more detailed plan shown in Fig. 7.[2] The construct-react-plan plan constructs

---

[2] Note that the plan description we use here is hierarchical; a particular action in one UML activity diagram may itself be described using a more detailed diagram.
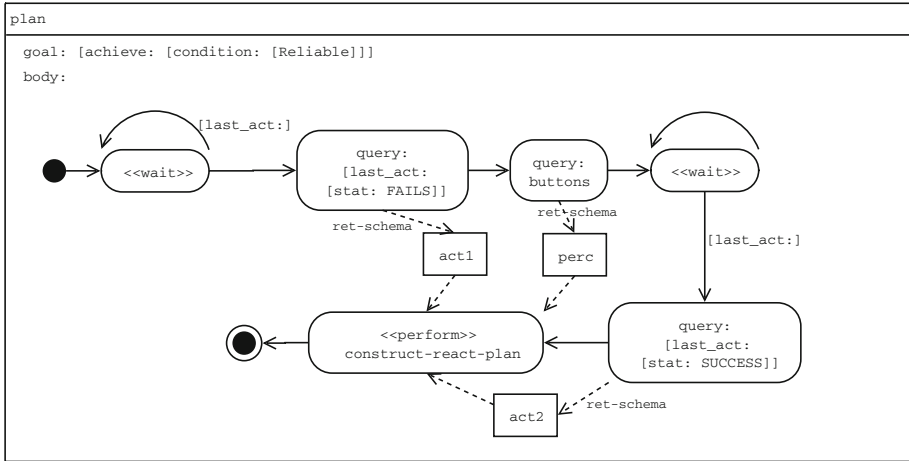
Fig. 6 Learning plan for learning a simple condition-action structure of plan. The `construct-react-plan` perform action calls a subplan presented in Fig. 7
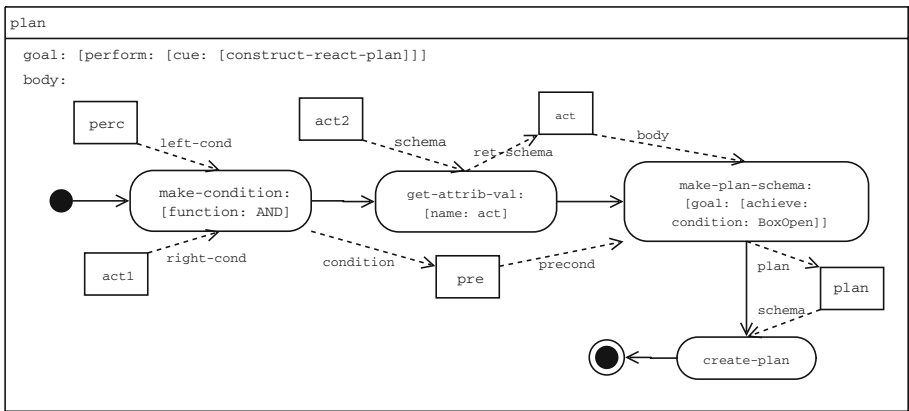

Fig. 7 Subplan for constructing a simple condition-action structure of plan that is called by the learning plan in Fig. 6

a new condition-action plan in a straightforward manner. A sequence of actions composes the parts of the new plan by putting the first failed action (`act1`) and the perception (`perc`) as the condition in the precondition attribute of the new plan. The successful action (`act2`) is then put in the body of the plan. The template of the plan that can be produced by the construction plan is illustrated in Fig. 8.

Intuitively, using the learning plan in Fig. 8, the rat agent hypothesizes that a single step of action (`act2`) will bring about the reward when a certain condition (`perc`) is perceived and the agent has just performed certain action (`act1`).

Another learning plan for the reactive structure creates condition-action structure but with an intermediate state (see Fig. 9). Extending the idea described in Fig. 6, the new plan body uses conditional branches to respond to the condition of the status of domain level plan once the second action is identified. In JAM, the conditional branches (or OR structure) means that the sequence of actions is executed and evaluated in a consecutive order. second branch
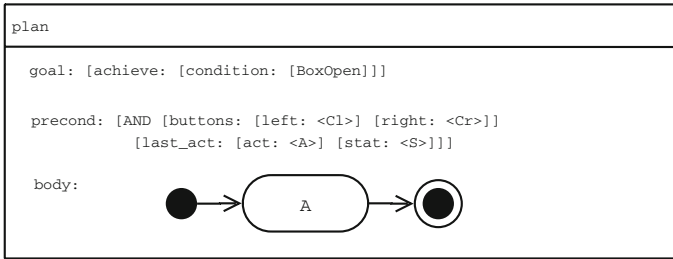
Fig. 8 The template of a plan constructed with the simple condition-action learning plan in Fig. 6
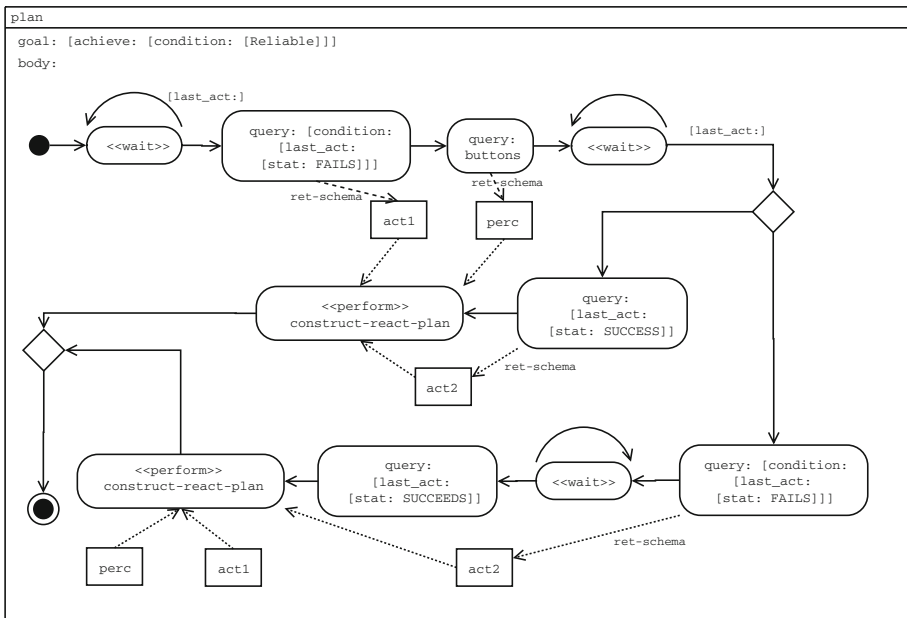

Fig. 9 Learning plan for learning simple condition-action structure with intermediate state

will be executed only if an action in the first branch fails. The first branch will be pursued if the second action identified in `last_act` succeeds. Then, it just behaves like the plan in Fig. 6 that a condition-action structure will be created. However, if the second action fails, the second branch then tries to identify another action. If the second branch succeeds, an intermediate plan is constructed using the perform action `construct-react-plan` but the action to be put in the plan body is not last successful action. Instead, the second last action (`act2`) that failed is used in the plan body.

The intermediate plan is a plan that may lead the agent into an intermediate state before choosing the appropriate plan for achieving the goal. Two kinds of plans may be generated as shown in Fig. 10. A plan that is created based on the template in Fig. 10i leads an action A1 directly to the goal after action ⟨a1⟩ that failed when a certain configuration of buttons was perceived. On the other hand, a plan that is created with the template in Fig. 10ii leads an action A2 to a state in which the plan in template (i) is applicable, after the agent performs another action ⟨a2⟩ which might also fail when a certain configuration of buttons was perceived.
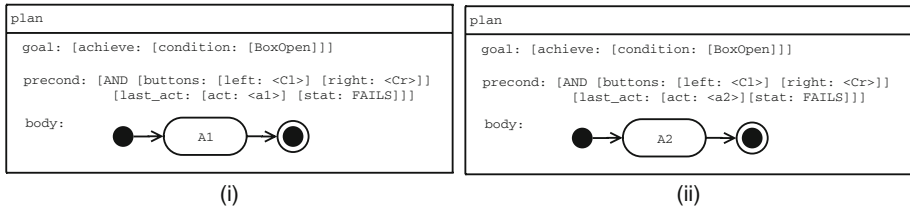
Fig. 10 The template structure of a plan constructed with the condition-action with intermediate state learning strategy in Fig. 9
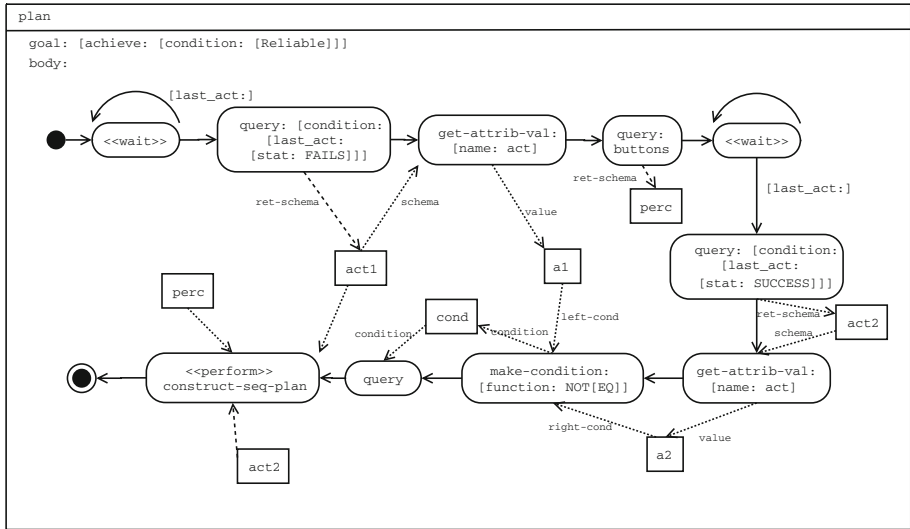

Fig. 11 Learning plan for learning complex structure (sequence). The `construct-seq-plan` perform action calls a subplan presented in Fig. 12

The learning plan for complex actions structure is similar to the above reactive learning plans. The difference is on how the target plan is constructed and the information to be put into the target plan's attribute. Figure 11 shows the learning plan for constructing plans with a sequence of actions. After an action fails and gets stored in `a1`, the second attempt to be found is a success attempt that is stored in `a2`. This pattern of consecutive actions identification is a reminiscent of the two last learning plans described above. However, a further check is performed to make sure the two consecutive actions are different using the `query` action. After that, the new plan can be constructed with the perform action `construct-seq-plan`. This process of acquiring a sequence plan means also that the hypothesis is that the goal can be achieved by performing a series of two different actions consecutively after perceiving a certain configuration of buttons.

The subplan for creating the sequence is shown in Fig. 12. The plan construction process is also straightforward. The perception of the buttons configuration `perc` is put as the precondition. The first action `a1` is then put into the plan body. Next, the action `a2` is appended after `a1` in the plan body before the whole plan is stored in the plan repository.

The other variation of learning a sequence is the one which includes the information of the past action as parts of the target plan precondition. This requires the learning plan to query the last action before capturing the consecutive actions that are successful at the end.
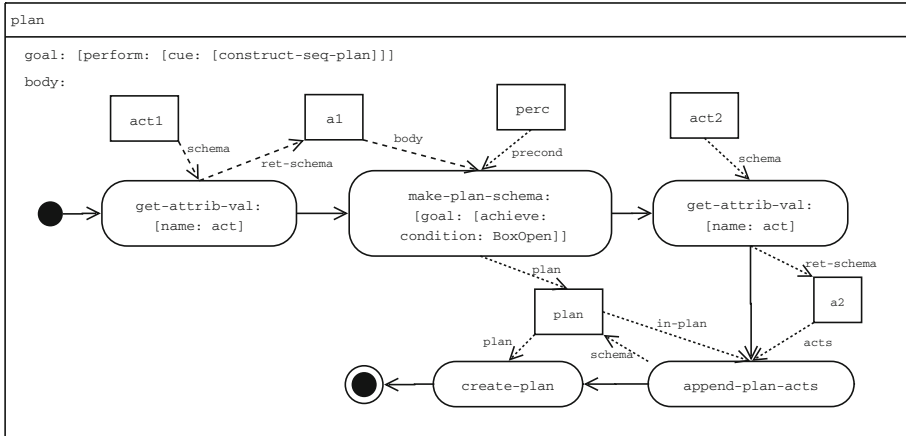
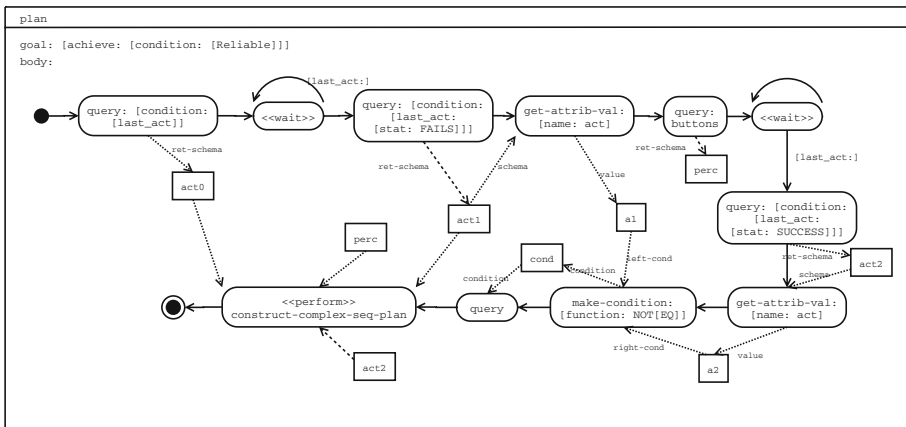**Fig. 12** Subplan for constructing a sequence of actions that is called by the learning plan in Fig. 11

Fig. 13 Learning plan for learning complex structure (sequence) with information about the past action. The `construct-complex-seq-plan` perform action calls a subplan presented in Fig. 14

Figures 13 and 14 show respectively the learning plan and the subplan for plan construction that learn a sequence with information about the past action. Although the rest of the steps are the same as those in Fig. 11, the learning plan in Fig. 13 begins with a query to an action before the wait action is performed. The result of the query is stored in `act0`, which refers to the action that may be preceding the sequence. The constructing plan in Fig. 14 puts `act0` in the precondition together with the perception, instead of inside the plan body.

Both learning plans for constructing complex actions structure (sequence) produce plans based on the schema in Fig. 15.

4.1.2 The experiment and the results

The method used in the rat's world experiment is to compare and evaluate different configurations of learning. Each single learning configuration is characterized based on two
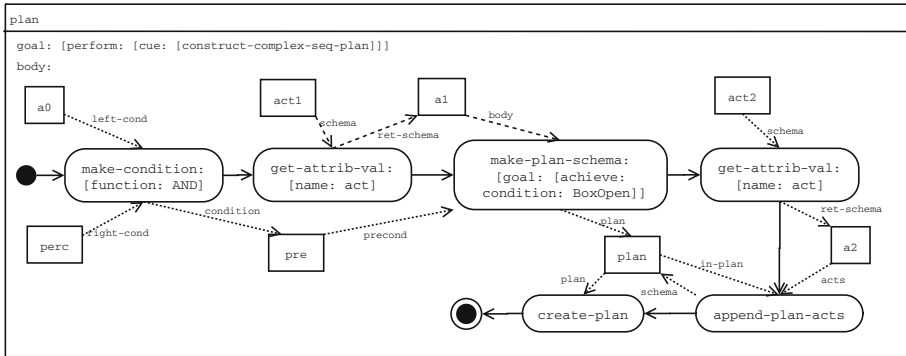
Fig. 14 Subplan for constructing a sequence of actions with information about the past action that is called by the learning plan in Fig. 13
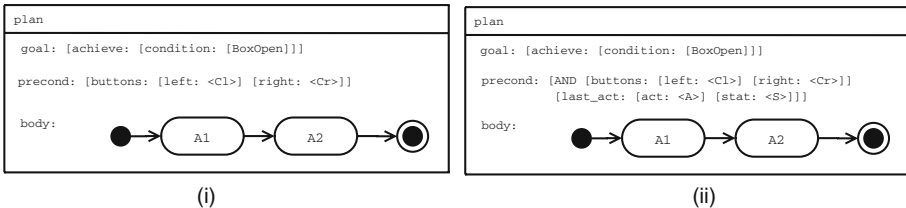

Fig. 15 The template structure of a plan constructed with the complex actions structure (sequence), (**i**) with a simple observational precondition which is produced by the learning plan in Fig. 11; (**ii**) with additional information about the past action which is produced by the learning plan in Fig. 13

parameters: the environment condition and the learning plan. There are two types of environment condition used in the experiment: (A) two buttons with fixed positions; (B) two buttons with changing positions. In the latter, buttons might swap their position when the rat performs a certain combination of actions. Following are the leaning plans we tested (see above for more detail):

- Learning 0. This plan does not apply learning or the decisions made are purely random.
- Learning 1. This learning plan, which is shown in Fig. 6, is used to construct a simple condition-action plan pattern.
- Learning 2. This learning plan, which is shown in Fig. 9 is used to construct a condition-action type of structure with an intermediate plan.
- Learning 3. This learning plan, which is shown in Fig. 11 is used to construct a sequence structure of plan.
- Learning 4. This learning plan, which is shown in Fig. 13 is used to construct a sequence structure with some reflection on the past action.

A particular learning configuration is named with its corresponding learning strategy and the type of environment. For example, learning 1A refers to a configuration in which the learning strategy for the simple condition-action is applied to the environment with the position of the buttons unchanged. Whereas learning 3B refers to the situation that the agent uses sequence generating strategy in the changing buttons situation.

As mentioned before, the performance of the agent is measured by the formula $p = 2S/T$ which takes into account the number of successful attempts (S) over a certain period of time (T). This measure also determines how and when the learning is conducted. For all config-
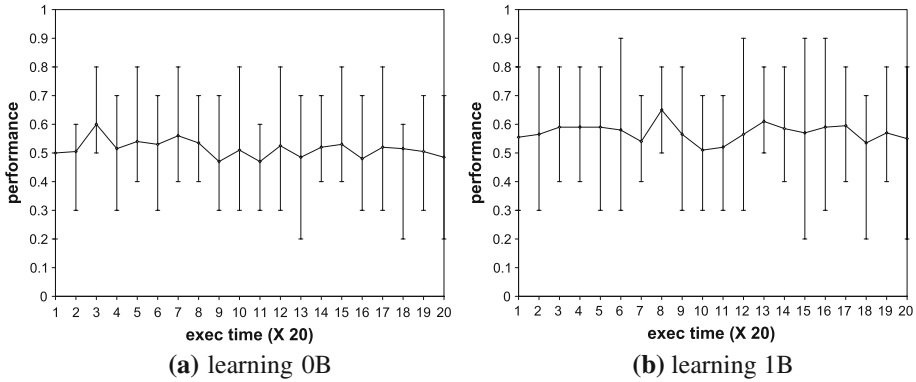
(a) learning 0B   **(b)** learning 1B

Fig. 16 Average and max–min range of performances of learning 0B and 1B over 20 × 20 (400) execution cycles

urations, T is set to 20 which means that the performance value is sampled and measured every 20 execution cycles. Each configuration runs for 400 execution cycles and there are 20 running instances for each configuration so that the variation of the learning outcomes can be captured.

Experimental results have shown that learning a simple condition-action structure in a changing condition (configuration 1B) is marginally better than no learning at all, which is equivalent to a random behavior (0B). In Fig. 16a, the mean value (of the 20 trials) of the task performance without learning (0B) is steady around half of the range (0.5) but with a range of variation which occasionally reaches minimally 0.2 and maximally 0.8. The mean value (20 trials) of the performance using the simple rule learning (1B) in Fig. 16b is steady around a slightly higher value (around 0.55) with a range of variation that seems to be shifted up which occasionally reaches minimally still in 0.2 but maximally 0.9.

In contrast, the learning configuration 3B which is learning for a sequence structure significantly outperforms both the random decision and the learning for simple condition-action structure in the dynamic situation. In Fig. 17a, the mean performance increases steeply and converges to the optimal value (1) in about half time of the whole execution period. Although there are some variations of value in different trials with a range of 0.2 and 1, all of them
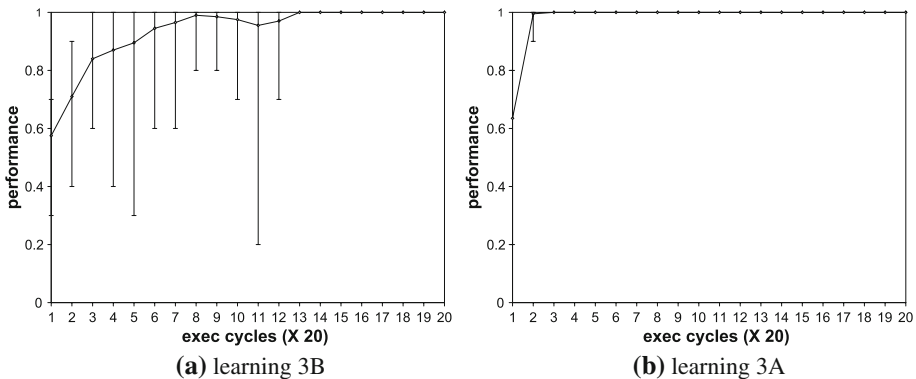

(a) learning 3B   **(b)** learning 3A

Fig. 17 Average and max–min range of performances of learning 3B and 3A over 20 × 20 (400) execution cycles

just stick with the optimal one after 260 (13×20) execution cycles. It seems that the performance levels off and stay on the optimum value. The fact that learning strategy 3 has a significant effect to the performance of the agent is strengthened by the result from the same learning strategy in an unchanging situation (configuration 3A) which is shown in Fig. 17b. The performance reaches the optimal value more abruptly with much less variation. The first stage of the experiment confirms the predisposition that the learning strategy for constructing complex structures like sequence (strategy 3) can be predominantly more effective to gain performance even in a dynamic situation.

These results have revealed that a certain learning strategy based on a hypothetical complex structure of actions which exploits rich plan representation can have a significant effect on the overall performance and the behavior of the agent. A deeper analysis through the data and the traces of execution reveals that the plan produced by learning 1 can not represent the appropriate structure of actions. Meanwhile, learning 3 can create appropriate plans to cope with the rule of the environment even though the situation is changing.

The condition of learning with the same strategy in the unchanging situation (1A) also indicates the limitation of representation to cover the appropriate knowledge produced by the learning plan. Figure 18 shows the comparison between the random decision strategy (0A) and the simple condition-action learning (1A) in the unchanging situation. The mean performance and the max-min variation of learning 1A in Fig. 18b are entirely higher than the learning 0A in Fig. 18a. However, performance still doest not converge to the optimal value even though it occasionally reaches the top value [like the maximum performance at time point 160 (8×20)]. A deeper analysis through the data and the traces of execution has revealed that although the highest performance can sometimes be obtained, the chance to get it is still low. Learning the right plan is sensitive to the order of the hypotheses produced. In other words, The number of possible hypotheses to make is large but there is actually a few number of plan that suits the condition of the task environment. There is a lesser chance that the agent finds the right combination of actions although it is still possible.

The learning plan 1 can be modified so that some intermediate plan may be learnt to bridge the gap in the representation capability . This approach is used in the learning plan 2. Figure 19 shows the results from the use of learning plan 2 in both unchanging and changing situations. Both situations (learning 2A and 2B) produce much higher performances than learning plan 1. The performance of learning 2A (Fig. 19a) increases and fluctuates around
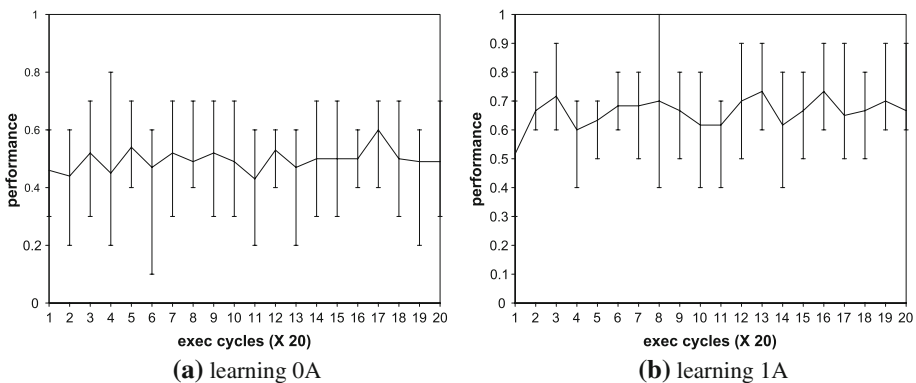

(a) learning 0A

(b) learning 1A

Fig. 18 Average and max–min range of performances of learning 0A and 1A over 20×20 (400) execution cycles

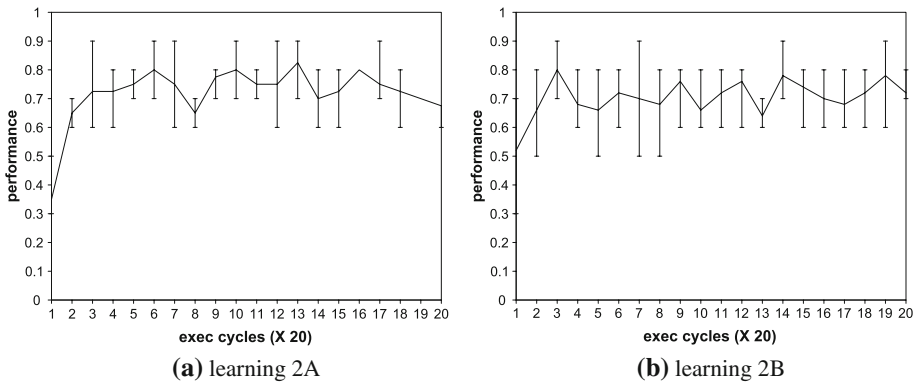**(a)** learning 2A   **(b)** learning 2B

Fig. 19 Average and max–min range of performances of learning 2A and 2B

some high values with a range of variation from 0.6 in minimum and maximum 0.9. The learning 2B (Fig. 19b) has a similar range of value and variation.

An analysis through the data and the traces of execution also shows that although the performance is relatively higher than learning 1, the chance of obtaining it is still lower than with learning 3. This indicates that there is still a flaw in the structure of plans produced by learning plan 2 in order to acquire the optimal knowledge.

The next learning plan investigated is learning 4 which is the combination of learning a sequence structure with more information about the past action in the precondition. Quite surprisingly, Fig. 20 shows that the additional information in the precondition of the learnt plan has a negative impact in both dynamic and unchanging situation. In the unchanging configuration (4A) in Fig. 20a, the mean value stays considerably high but in many cases it still does not converge to the optimum. There are only two instances which are found to eventually reach the optimum. On the other hand, using the same strategy in changing situation (4B) exhibits different patterns from its static counterpart. The mean performance is high, but it never reaches the optimum.

A deeper analysis through the execution traces shows that in the configuration 4A it is possible that the optimal plan can be learnt. However, it must be in the condition that a plan
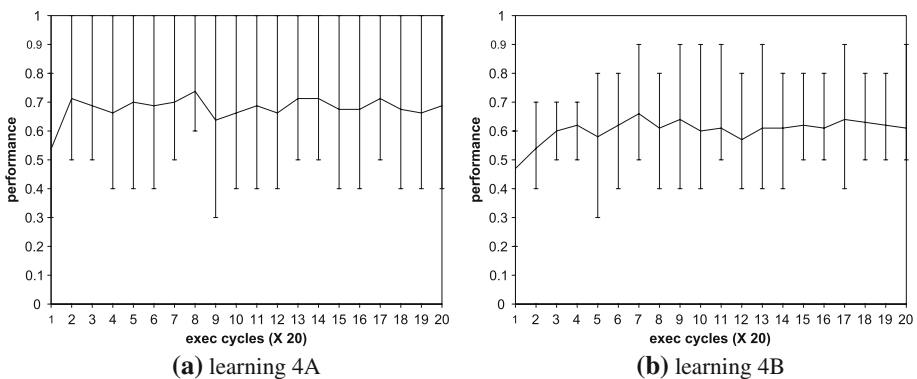

(a) learning 4A   **(b)** learning 4B

Fig. 20 Average and max–min range of performances of learning 4A and 4B over 20×20 (400) execution cycles

consisting of a precondition that refers to a previously successful attempt is firstly learnt. This condition is not frequent and is mostly preceded with some failures. On the other hand, in the configuration 4B in which the situation is changing, the chance to learn the optimum plans is even smaller. It is possible that at one time a plan with a reference to the last success can be learnt, but to learn all the necessary plans for the optimal solution, the learning still needs more time. There must be a situation that a successful plan in the last attempt is learnt without any failed plan is learnt preceding that. However, it rarely happens that those successive learning can occur.

From all configurations of the learning task in the rat's world experiment, only those that apply learning plan 3 can learn the optimal solution. Other plans still can not learn the right knowledge. The capability to learn the optimal plans is also sensitive to the order of which pattern is learnt before another.

In summary, the experiment in the rat's world reveals some important facts about the intentional learning architecture, which can be summarised as follows:

- A learning plan that produces a rule-like condition action structure, even with the inclusion of the reflection of the past action status, may still have some gaps in the coverage of the knowledge structure to acquire the optimal solution. The gaps exist especially when the last success or positive reward is not taken into account in the learning process.
- A simple learning plan that uses only little information about the environment, but creates a more complex structure of action than just merely a condition-action structure, can acquire the optimal solution as long as the structure fits with the rule of the environment.
- Additional information can sometimes be counterproductive and impede the process of acquiring the knowledge structure. Much information captured are unnecessary so that the chance of producing the right hypothesis is lower.

4.2 Vacuum explorer

The second domain called the vacuum explorer, is used to further investigate aspects of different hypothetical structures of actions produced by different learning plans when the agent continuously tries to achieve its objective. In this domain, the agent explores and navigates around a two dimensional grid environment. Its main task is to find and collect as many target objects as possible. To consume a target object, the agent can just go through it and the target will just disappear from the environment.

The vacuum explorer is more complicated than the rat's world because the agent has a very limited perception and some states of the environment may be hidden from the agent's beliefs. The agent can only sense locally around its location up to a very limited distance. To navigate effectively around the environment the agent must learn when and how to move based on information given in local situation and some past experiences.

As a testbed, the environment of the vacuum explorer can be programmed and configured to enforce different patterns. In particular, one or many objects can be put in a grid-based world like in Fig. 21. The kinds of object that can be put in the environment are the agent itself, passive obstacles that block the agent's movement, and targets. The designer can control the number of targets, and whether they reappear after consumption.

The agent can move around the space by a limited number of primitive actions as follows:

- `move-forward`. This action makes the agent move a step forward in a grid. The direction of movement depends on the current heading of the agent. The action fails if there is an obstacle just in front of it.

Fig. 21 The vacuum explorer: navigating around a grid world
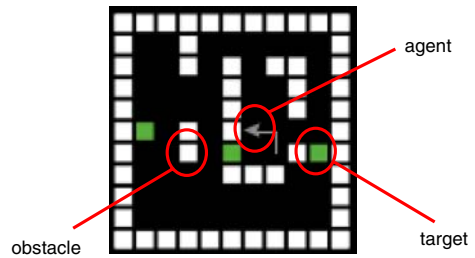
- turn-left. This action makes the agent turns 90° counter-clockwise and a step forward after that.
- turn-right. This action makes the agent turns 90° clockwise and a step forward.

The agent can sense surrounding objects at one distance unit to the left, front left, front, front right, and right side of the agent. The perception of the agent can be expressed as a schema

$$[\texttt{sense-buff} : [\langle o_1 \rangle \langle d_1 \rangle][\langle o_2 \rangle \langle d_2 \rangle][\langle o_3 \rangle \langle d_3 \rangle][\langle o_4 \rangle \langle d_4 \rangle][\langle o_5 \rangle \langle d_5 \rangle]]$$

Both o_n and d_n denote the type of object and the distance from the agent respectively. The index n points to the relative direction from the agent. The order is from 1 to 5 which corresponds to the relative positions from the left side to the right side if turning clockwise. If there is no object at a particular sensing position, the type o_n is filled with a dot symbol (.) with a distance $d_n 0$.

In terms of the intentional learning model and the evaluation of learning plans, the layout reflects the possibility of knowledge structure that can be learnt by the agent. For example, a layout of obstacles that forms a passage way or a corridor reflects a plan to follow a straight line or a wall, whereas a layout with too much blank space or unstructured obstacles may reflect interchanging actions (like zigzag movement) or a random sequence. The testbed is used to explore if a learning plan can produce some useful knowledge for the agent to navigate around the environment.

4.2.1 The implementation and learning plans

Similar to the rat's world, the vacuum explorer simulation is implemented using the Netlogo multi-agent modeling environment [40]. However, a new BDI interpreter was also implemented in Netlogo. This is due to the limitation in existing BDI agent platforms. Most existing BDI implementations lack the feature of runtime plan modification. Although some frameworks have supported that feature (e.g. [19,29]), they are still lacking in the capability to capture the state of intention at runtime. In dealing with more complex situations and complex structures of knowledge, specialized interpreters like that of JAM platform proved limited.

On the other hand, though it is not designed to support complex symbolic processing, the Netlogo supports rich list structure manipulation which is enough to implement a prototype of the intentional learning agent model. The agent reasoning engine can capture the state of the intention structure on the run, and accordingly change the plan library. Consequently, it does not have to use the concurrency between the domain and the learning intention (as we had to do in the rat's world experiment). The learning can be made as an intention that controls the execution of another intention at a subordinate level.

As parts of its learning strategies, the vacuum explorer also uses negative plans as low level domain plans created to avoid the adoption of a bad plan when the same situation occurs. A negative plan is created in the plan library with a single action structure in its plan body. It has the condition-action structure with the observation put as the precondition. However, the utility of this plan is negative and it is created when the action expressed in the plan body has failed when taken from the same condition as expressed in the precondition. This plan will be used to filter out and reject options that have the same action as the body of a negative plan and occur in the same situation as described in the precondition of the negative plan. The use of negative plans enhances the speed of the learning and the deliberation process. The template structure of the negative plan is shown in Fig. 27ii. Another feature that is not available in its domain counterpars (the rat's world) is that a learnt plan in can still be removed when it is considered to be less useful or no so successful. The utility update mechanism may update the utility attribute of a plan and when it reaches a negative value, a separate process will remove it from the plan repository. However, this plan deletion process is not applied to the negative plans as their removal would instead inhibit their function of preventing failures.

The learning goal in the vacuum explorer is activated when the default operation has failed more than 10 times. By default, three different primitive plans are used to navigate around the environment. One is for moving a step forward, one is for turning the agent counter clockwise in 90°, and the other is for turning 90° clockwise. Figure 22 shows the default plans in the vacuum explorer domain.

There is also a special-purpose plan which is dedicated only for adjusting the utility of the learnt plans that is applied in all those three learning plans. The plan for adjusting utilities of constructed plans is presented in Fig. 23.

The utility adjustment plan is activated by the goal condition to achieve `learn-option` which is posted by the deliberation cycle when the number of failures has exceeded a certain limit. This learning goal is also used to trigger other learning plans. The source actions `in-body`, the source goal `in-goal`, and the source plan (`in-plan`) are the parameters for the learning plan as the components of the agent's performance to be improved. These information are provided by the main execution cycle when the level of performance of an intention drops or is still under a minimum threshold. The source actions `in-body` is monitored using the `monitor` action to check if the actions fail or not. The next step is to deliberate and select a new plan through the `deliberate` action given the source goal `in-goal`. The source plan, the result of executing the actions, and the new selected plan are
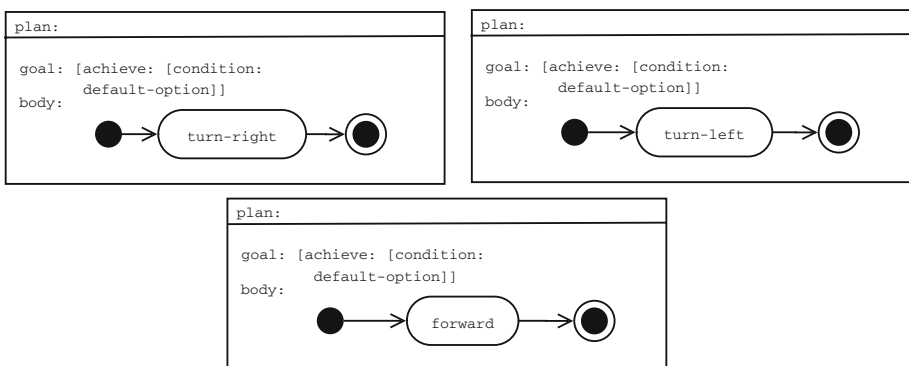

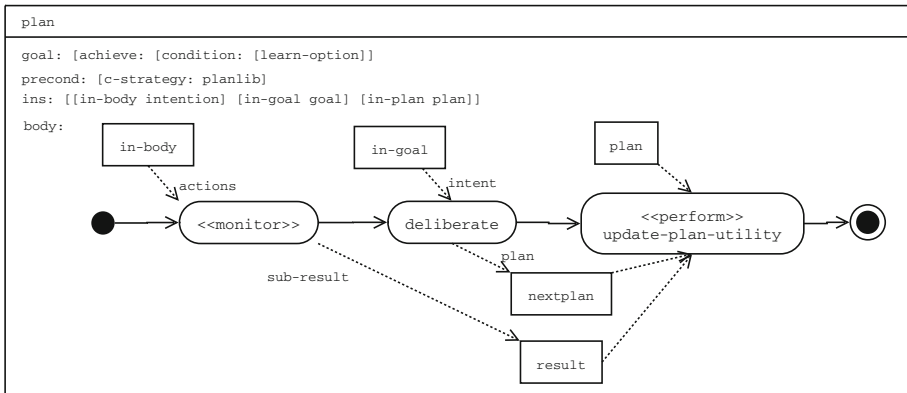**Fig. 22** Default plans in the vacuum explorer domain

Fig. 23 Updating learnt plan utility. The perform update-plan-utility calls a subplan shown in Fig. 24
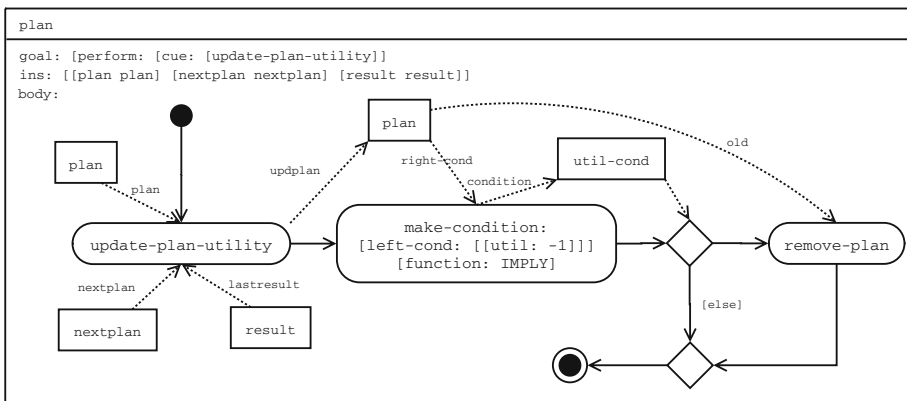

Fig. 24 Subplan for updating a plan utility which is called by the learning plan in Fig. 23

used for adjusting the utility of the plan. The utility updating process is described in detail in Fig. 24 as a plan for the perform subactivity of update-plan-utility in Fig. 23.

In the perform subplan update-plan-utility, an action operator specifically built for updating the utility attribute of a plan is used. The action has the same name as its plan cue that is update-plan-utility action which is depicted in Fig. 24. The action uses three inputs plan, nextplan, and lastresult as the current selected plan to be updated, the next plan that is selected from the deliberation process after execution of the first plan, and the result of the execution of the plan respectively. The utility updating action changes the utility of the plan based on the utility of the next plan selected in the next deliberation. The utility of the next plan is propagated to the current plan with a reduction of value 1. If the the plan execution fails, then the utility is just substracted by 1. In the utility updating plan, when the utility of the plan to update is negative, then the plan is just removed from the plan library. This approach of utility propagation is reminiscent of Q-learning as a reinforcement learning algorithm [39].

The first type of learning plan used in vacuum explorer domain is the strategy that creates the plans with a rule-like condition action structure. This learning plan is presented in Fig. 25. This learning plan creates a hypothetical plan based on the observation and the
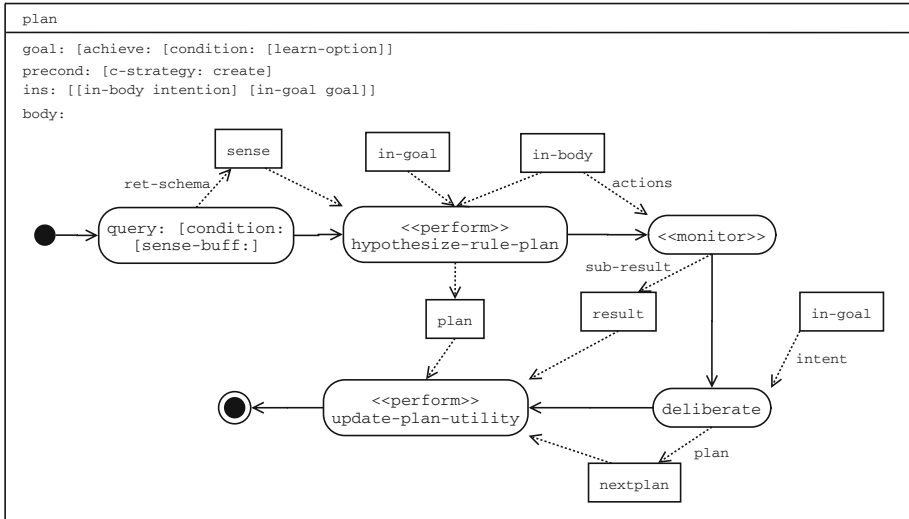
Fig. 25 Learning plan 1: making rule-like plans

current selected intention. The created plan is then tested using the monitor action and its utility modified using the `update-plan-utility` subplan. As in the plan for updating utility in Fig. 23, the plan receives and uses input parameters like `in-body` and `in-goal` as the source actions and the source goal respectively. This plan also match and select a plan from the plan library by `deliberate` action. However, instead of taking the source plan and updating its utility, a new plan schema is created, based on the actions contained in `in-body`, the goal described in in-goal, and sense as the current perception obtained by the query action. This learning plan is shown in Fig. 26. This special plan can produce a plan structure which can be depicted in Fig. 27i.

The second type of learning plan constructs a repetition structure. This plan (shown in Fig. 28) creates a hypothetical plan that contains a repetitive structure in the plan body using
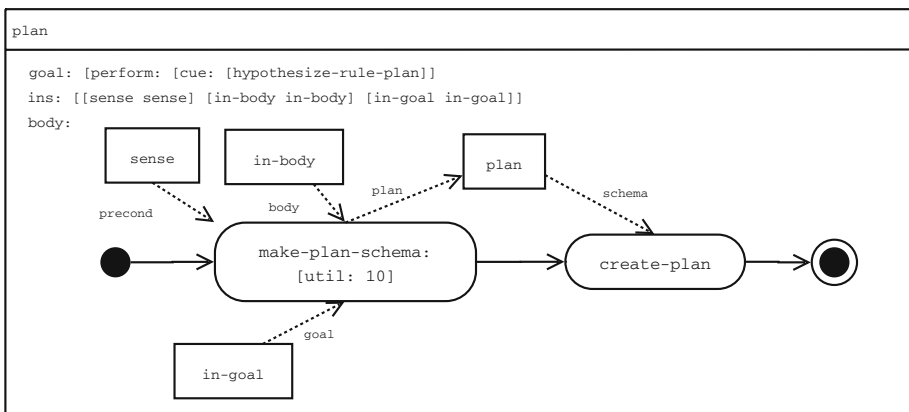

Fig. 26 Subplan for creating the hypothetical condition-action structure of plan which is called by the learning plan in Fig. 25
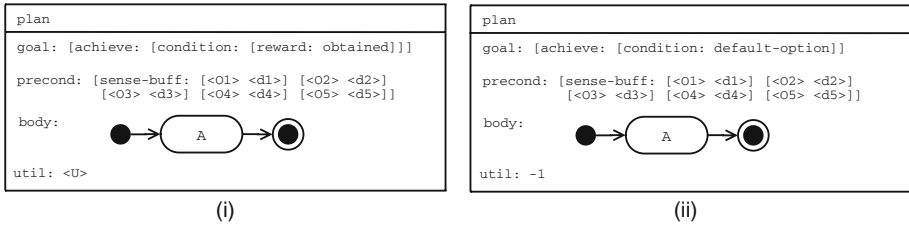
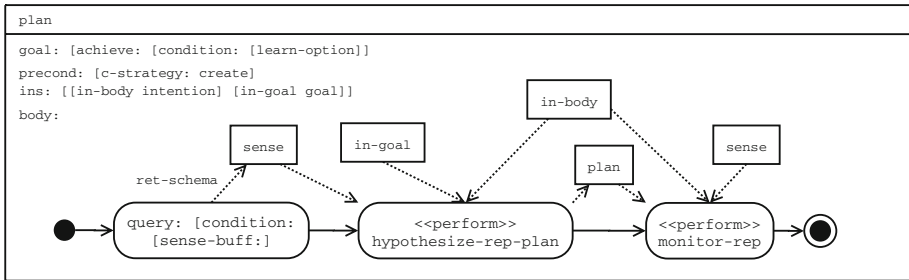**Fig. 27** The template structure of a plan constructed with the simple condition-action learning plan in Fig. 25

Fig. 28 Learning plan 2: making repetition plans
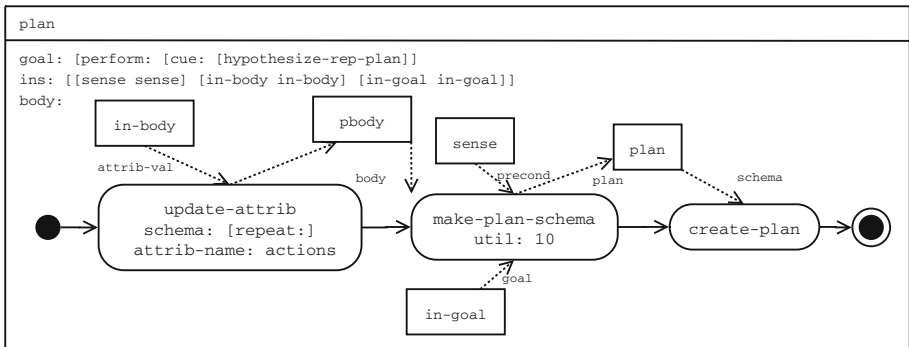
Fig. 29 Subplan for creating the hypothetical repetition structure of a plan which is called from the learning plan in Fig. 28

the perform action `hypothesize-rep-plan`. The created plan is then tested iteratively inside the perform action `monitor-rep` for building up the repetition.

The plan `hypothesize-rep-plan` for constructing the hypothetical repetition plan schema is presented in Fig. 29. The plan can produce a plan structure which is illustrated in Fig. 33i. The `hypothesize-rep-plan` creates a new plan schema by composing the perception in `sense`, the goal in `in-goal`, and the plan body in `pbody` consisting a repetition structure with the action `make-plan-schema`. The plan body `pbody` is built by the action `update-attrib`. The new created plan schema initially has no limit of iteration and the utility attribute is set to 10. The value of the utility attribute can be an arbitrary positive number. However, the utility value can actually determine the number of iteration which is described in detail latter.
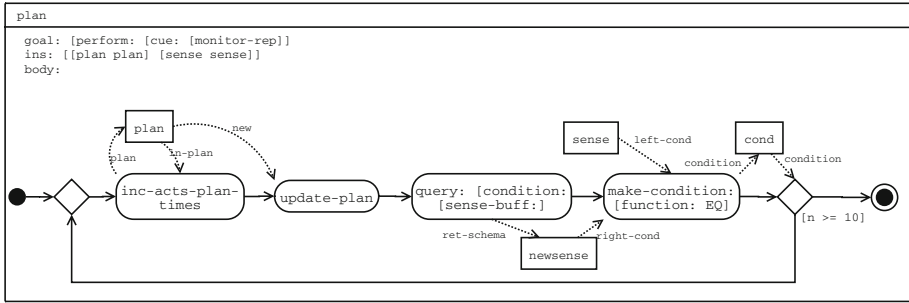
Fig. 30 Subplan for building up a repetition which is called by the plan in Fig. 28

The important part of the this second type of learning plan is the plan for monitoring and building up the repetition in the perform action `monitor-rep` which is presented in detail in Fig. 30. The `monitor-rep` plan updates (increases) the repetition limit while testing the action for a failure or a change in the agent's perception iteratively. The cycle stops when the agent get a different perception from its previous observation by the assumption that a different observed situation may require the agent to change its plan. The monitoring process for generating repetition also stops if the number of iteration exceeds 10 which means also that the utility of the current plan would be less than zero. The `inc-acts-plan-times` executes the monitored action and increases the iteration limit in the repetition structure of the plan body of `plan`. The query action gets the current perception, and if there is no difference in the current perception from the previous one and the number of iteration is still not exceeding 10, then the process is repeated from the beginning. Otherwise, the process stops with the new plan in the plan library.

The third type of learning plan constructs a sequence structure of actions in the plan body. The learning plan is presented in Fig. 31. Similar to the plan for creating repetitions, this learning plan creates a hypothetical plan that contains a sequence structure in the plan body. The created plan is then tested iteratively using the `monitor-seq` subplan for monitoring a sequence. The learning plan takes the actions (`in-body`), the goal (`in-goal`), and observes the perception (`sense`), to create a new plan and test it on the run.

The plan for constructing the hypothetical sequence plan (`hypothesize-seq-plan`) is presented in Fig. 32. The plan can produce a plan structure which can be illustrated in Fig. 33ii. The important part of the this third type of learning plan is the subactivity for moni-
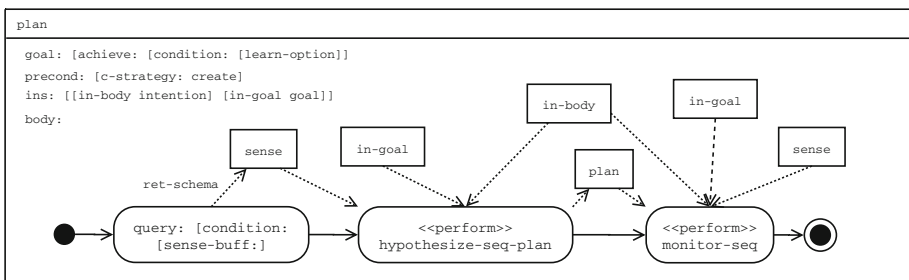

Fig. 31 Learning plan 3: making sequence plans
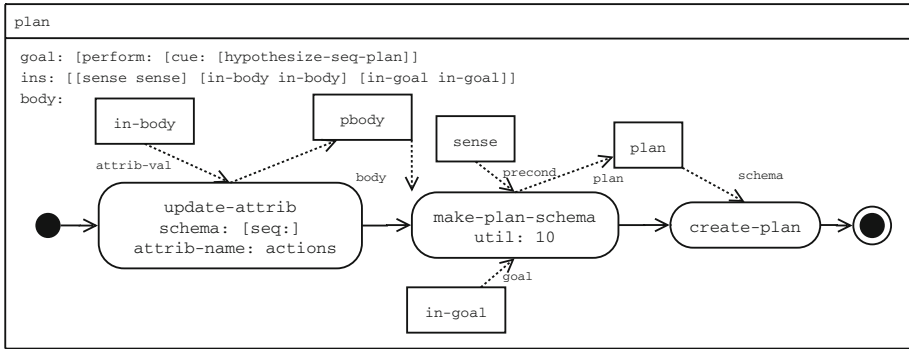
Fig. 32 Subplan for creating the hypothetical sequence structure of a plan which is called by the plan in Fig. 31
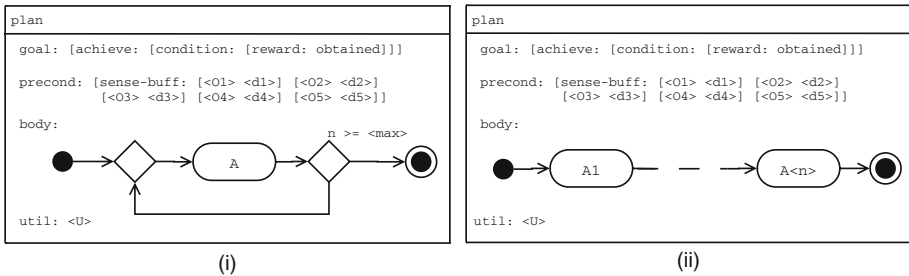
Fig. 33 The template structure of a plan constructed with the (**i**) repetition structure created by the process in Fig. 29 and (**ii**) sequence structure by the process in Fig. 32
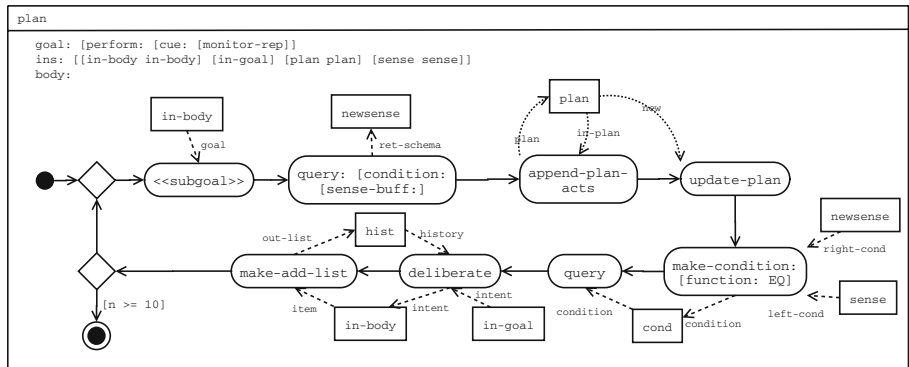
Fig. 34 Subplan for building up a sequence which is called by the plan in Fig. 31

toring and building up the sequence in the perform action `monitor-seq` which is presented in detail in Fig. 34.

At the beginning the `monitor-seq` plan tests the action using the meta action `sub-goal`. This `subgoal` tests if the monitored action succeeds. Otherwise this `monitor-seq` plan fails which also causes the overall learning plan fails. After testing the action a new perception is observed using the `query` action and is stored in `newsense` and the tested action is appended in the body of the plan by `append-plan-acts` action. The `monitor-seq`

plan also checks if it still perceives the same thing as before after executing the action. The `query` action checks the difference. When the current perception and the previous one are the same, then the learning plan fails, although the plan learnt so far is still retained in the plan library. This contrasts with the repetition strategy that always proceeds as long as the observation is the same. The intuition behind the sequence generation is that each subsequent action should be different from one another. Otherwise, the repetition structure is more suitable.

However, unlike its repetition counterpart, the `monitor-seq` plan is always looking for a different step of action. The `deliberate` action uses historical data from `hist` to select a new different action or plan. The selection also updates the history `hist` using `make-add-list` action. The same with the repetition learning plan, the limit of the number of actions in the sequence is also 10.

4.2.2 The experiment and the results

The main purpose of the vacuum explorer domain is to find some patterns of activity of the agent involving the intentional learning process that indicate the acquisition of useful knowledge for the agent to achieve its goal. The approach used in the vacuum explorer is by comparing the results of experiments from different configurations of learning situation. However, the comparison is used only for detecting some interesting evidence in which further examinations are focused to search for some useful patterns. Firstly, the comparison is conducted by looking at the number of targets the agent can get over a period of time for each learning configuration. In this stage, the number of plans learnt is also evaluated besides the cumulative achievements. Secondly, from the comparison of targets achieved and the plans created, several cases indicating the reuse of learnt knowledge over high achievement are selected and analyzed further by looking at how plans are formed and changed, in relation to the number of achievements over certain periods of time.

Similar to the rat's world, each single learning configuration is characterized based on two parameters: the environment condition and the learning plan. The environment condition is based on two factors: the layout structure of the environment which is based on 11×11 2-dimensional grid and the number of targets available at one time. We experimented with three different layout structures in the vacuum explorer environment which are shown in Fig. 35: TUNNEL, TURNINGS, BARRIERS and the number of targets at a time to be explored are 1, 3, and 5.
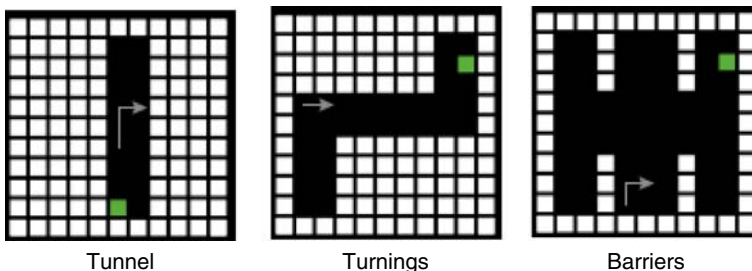

Fig. 35 Types of environment in vacuum explorer domain

Four different types of learning plans labeled as follows:

- Learning 1. This type of learning plan, which is shown in Fig. 25, is used to construct a condition-action plan pattern.
- Learning 2. This type of learning plan, which is shown in Fig. 28, is used to construct a repetitive action structure of plan.
- Learning 3. This type of plan, which is shown in Fig. 31 is used to construct a sequence action structure of plan.
- Learning 4. This configuration is the combination of different learning plans above (learning 1, 2, and 3). One learning plan can be activated at one time and at another time a different learning plan can take over.

A particular learning configuration is named with its corresponding learning plan and the type of the environment condition. For example, learning 1-TUNNEL-3 refers to a configuration in which learning plan 1 is applied to the environment with the TUNNEL layout structure and there are three targets put in the environment at a time.

A running instance of the agent was conducted over a period of 3000 execution cycles. That number is assumed to provide enough time for the agent to explore all parts of the environment. However, we do not provide detailed analysis and results of the vacuum explorer experiment in this paper as it involves overwhelming analysis on the trace of executions and the plans produced. The reason is that the main purpose of the experiment domain is mainly to explore the characteristics of the application of different learning plans in different situations. It is not for evaluating and finding the best plan over another. In fact, the experiments have shown that each learning plan has its own beneficial feature on different environment configuration. Interested readers can refer to [38] for more information about the experiments and the analysis of the vacuum explorer domain.

The experimental results of the vacuum explorer domain have shown that the application of different learning plans can produce different performances and behaviors of the agent in the same environment condition.

Figure 36a shows the number of target objects collected by the agent over a period of time in the TUNNEL environment where there are three targets at a time. All learning plans can increasingly collect targets over 3000 cycles. The learning plan with the lowest rank is learning 2 which can get 32 targets in 3000 cycles. The next rank is learning 3 which reaches 55 in the same period of time. The highest rank 75 is held by learning 1, however the number of target reached is very close to the targets collected by learning 4 which is 74. Although both learning 1 and learning 4 can collect almost the same targets at the end, their rates of target achievements are different. Learning 4 seems to be ahead faster in the middle time points but can not find the goal for a while at the end. Learning 1 on the other hand, is much quicker in attaining the goals at the end (see below for more details).

Figure 36b compares the number of plans learnt by different learning plans over the period of time. It shows that the number of plan learnt in every strategy is changing most of the time. The change frequently occurs like pulsating signals because learning is creating hypotheses in the plan library and just deletes them later when they are invalidated. The most fluctuating learning processes, in terms of the number of plans acquired, is learning plan 2. From observation of the execution trace most plans learnt by the learning plan 2 are useless. Until the end of the time period, the number of learnt plan from learning plan 2 is practically one, though the contents are changed frequently. On the other hand, other learning plans, like plan 1, 3, and 4, somehow can retain their learnt plans. Plans that are learnt at the beginning of the execution are mostly kept in the plan library until the end of the experiment. This indicates
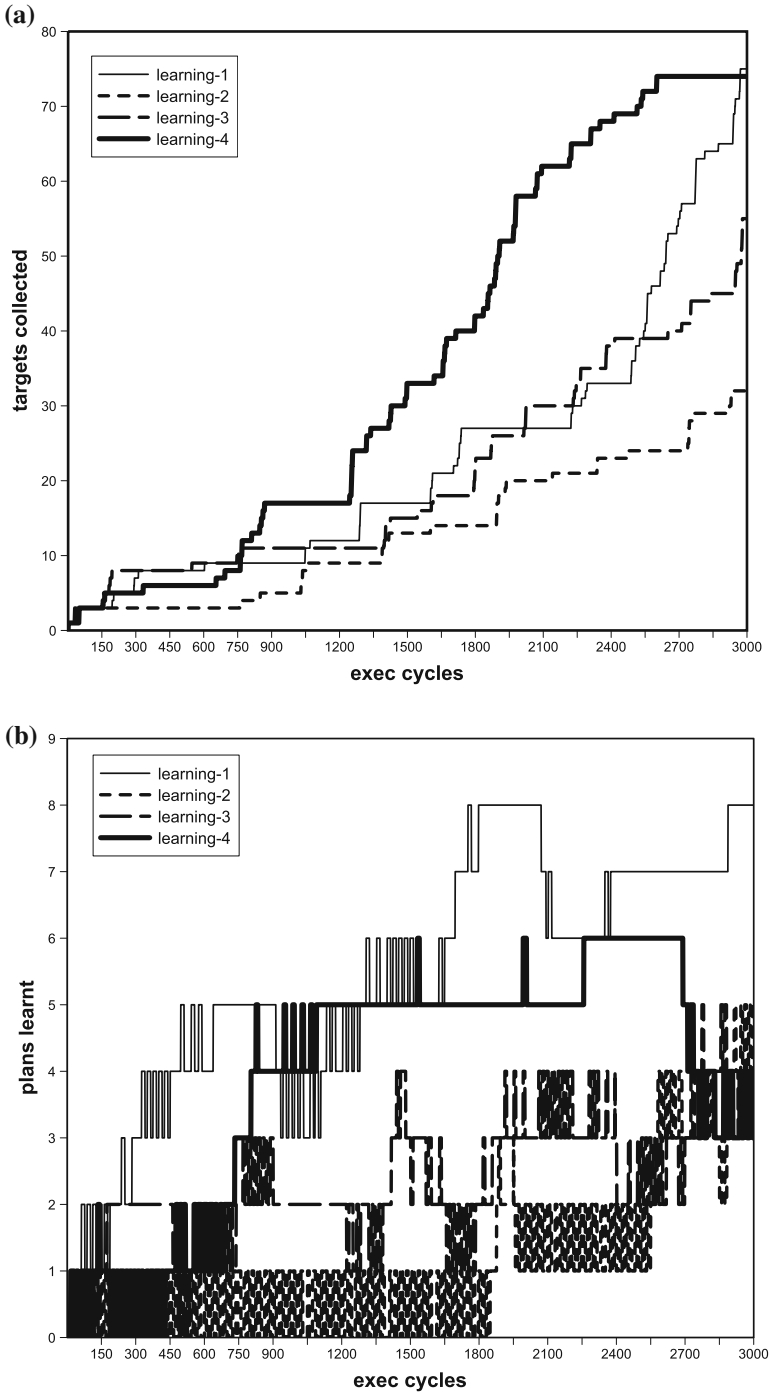
(a)

(b)

Fig. 36 Cumulative goals obtained and the number of plans learnt in the TUNNEL environment with target number 3. (**a**) Cumulative number of achievement. (**b**) Number of learnt plans

that most learnt plans from the start using learning plan 1, 3, and 4 are useful or, in other words, their utilities are high.

As an example of the plans produced, below are learnt plans in the configuration 1-TUNNEL-3 for certain time points. The goal attribute is hidden, as a simplification, with the assumption that the goal is the same for all the learnt plans.

Time point 707

```
[plan:
    [precond: [sense-buff [brick 1][brick 1][resource 1][resource 1][. 0]]]
    [body [do: turn-right]] [utility 10] [id created-1]]
[plan:
    [precond: [sense-buff [brick 1][brick 1][. 0][resource 1][. 0]]]
    [body [do: turn-right]] [id created-9] [utility 7]]
[plan:
    [precond: [sense-buff [brick 1][brick 1][brick 1][brick 1][. 0]]]
    [body [do: turn-right]] [id created-15] [utility 9]]
[plan:
    [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][.0]]]
    [body [do: turn-right]] [id created-7] [utility 9]]
[plan:
    [precond: [sense-buff [brick 1][brick 1][. 0][. 0][. 0]]]
    [body [do: turn-right]] [id created-19] [utility 8]]
```

Time point 1717

```
[plan:
    [precond: [sense-buff [brick 1][brick 1][. 0][resource 1][. 0]]]]
    [body [do: turn-right]] [id created-9] [utility 7]]
[plan:
    [precond: [sense-buff [brick 1][brick 1][resource 1][resource 1][. 0]]]]
    [body [do: turn-right]] [id created-1] [utility 9]]
[plan:
    [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][. 0]]]]
    [body [do: turn-left]] [id created-26] [utility 8]]
[plan:
    [precond: [sense-buff [brick 1][brick 1][brick 1][brick 1][. 0]]]]
    [body [do: turn-right]] [id created-15] [utility 9]]
[plan:
    [precond: [sense-buff [brick 1][brick 1][. 0][. 0][. 0]]]
    [body [do: forward]] [id created-32] [utility 9]]
[plan:
    [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][brick 1]]]
    [body [do: turn-left]] [id created-42] [utility 9]]
[plan:
    [precond: [sense-buff [. 0][resource 1][. 0][brick 1][brick 1]]]
    [body [do: turn-left]] [utility 10] [id created-43]]
```

Time point 2929

```
[plan:
    [precond: [sense-buff [. 0][resource 1][. 0][brick 1][brick 1]]]
    [body [do: turn-left]] [utility 10] [id created-43]]
[plan:
    [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][brick 1]]]
    [body [do: turn-left]] [id created-42] [utility 9]]
[plan:
    [precond: [sense-buff [brick 1][brick 1][resource 1][resource 1][. 0]]]
    [body [do: turn-right]] [id created-1] [utility 8]]
[plan:
    [precond: [sense-buff [brick 1][brick 1][brick 1][brick 1][. 0]]]
    [body [do: turn-right]] [id created-15] [utility 5]]
[plan:
    [precond: [sense-buff [brick 1][brick 1][. 0][. 0][. 0]]]
    [body [do: forward]]  [id created-32] [utility 6]]
```

```
[plan:
    [precond: [sense-buff [brick 1][brick 1][. 0][resource 1][. 0]]]]]]]
    [body [do: turn-right]] [id created-9] [utility 6]]
[plan:
    [precond: [sense-buff [. 0][. 0][. 0][brick 1][brick 1]]]
    [body [do: turn-left]] [utility 10] [id created-49]]
[plan:
    [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][. 0]]]
    [body [do: turn-right]] [id created-48] [utility 7]]
```

The learning plan 1 creates and accumulates plans that may lead the agent to the target. Some plans that are created from the early stage of the learning like plans with id `created-1`, `created-9`, and `created-15` (time point 707) still exist in later time points (1717). Some of the plans' utility values are changed which can be an indication that those plans are used and updated by the learning plan that modifies the plan utility.

With the plan configuration 4-TUNNEL-3, following are some sample learnt plans:

Time point 808

```
[plan:
    [precond: [sense-buff [brick 1][brick 1][brick 1][brick 1][. 0]]]
    [body: [do: turn-right]] [id created-18] [utility 9]]
[plan:
    [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][. 0]]]
    [id created-28] [body: [seq: [do: turn-left]] [utility 9]]
[plan:
    [precond: [sense-buff [. 0][. 0][. 0][brick 1][brick 1]]]
    [body: [do: forward]] [id created-29] [utility 9]]
[plan:
    [precond: [sense-buff [. 0][resource 1][. 0][brick 1][brick 1]]]
    [body: [do: turn-left]] [utility 10] [id created-30]]
```

Time point 1010

```
[plan:
    [precond: [sense-buff [brick 1][brick 1][. 0][resource 1][. 0]]]
    [id created-21] [body: [seq: [do: turn-right]]] [utility 9]]
[plan:
    [precond: [sense-buff [. 0] [brick 1] [brick 1] [brick 1] [. 0]]]
    [id created-10] [body: [seq: [do: turn-right] [do: turn-right]]]
    [utility 6]]
```

Time point 1414

```
[plan:
    [precond: [sense-buff [brick 1][brick 1][brick 1][brick 1][. 0]]]
    [body: [do: turn-right]] [id created-18] [utility 8]]
[plan:
    [precond: [sense-buff [. 0][resource 1][. 0][brick 1][brick 1]]]
    [body: [do: turn-left]] [utility 8] [id created-30]]
[plan:
    [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][. 0]]]
    [id created-28] [body: [seq: [do: turn-left]] [utility 8]]
[plan:
    [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][brick 1]]]
    [body: [do: turn-left]] [id created-36] [utility 7]]
[plan:
    [precond: [sense-buff [. 0][. 0][. 0][brick 1][brick 1]]]
    [body: [do: forward]] [id created-29] [utility 9]]
```

Time point 2424

```
[plan:
    [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][brick 1]]]
    [body: [do: turn-left]] [id created-36] [utility 3]]
```

```
[plan:
    [precond: [sense-buff [brick 1][brick 1][brick 1][brick 1][. 0]]]
    [body: [do: turn-right]] [id created-18] [utility 1]]
[plan:
    [precond: [sense-buff [brick 1][brick 1][. 0][. 0][. 0]]]
    [body: [do: turn-right]] [utility 10] [id created-39]]
[plan:
    [precond: [sense-buff [. 0][resource 1][. 0][brick 1][brick 1]]]
    [body: [do: turn-left]] [utility 0] [id created-30]]
[plan:
    [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][. 0]]]
    [id created-28] [body: [seq: [do: turn-left]] [utility 0]]
[plan:
    [precond: [sense-buff [. 0][. 0][. 0][brick 1][brick 1]]]
    [body: [do: forward]] [id created-29] [utility 0]]
```

Time point 2929

```
[plan:
    [precond: [sense-buff [. 0][resource 1][. 0][brick 1][brick 1]]]
    [body: [do: turn-left]] [utility 0] [id created-30]]
[plan:
    [precond: [sense-buff [brick 1][brick 1][brick 1][brick 1][. 0]]]
    [body: [do: turn-right]] [id created-18] [utility 9]]
[plan:
    [precond: [sense-buff [brick 1][brick 1][. 0][. 0][. 0]]]
    [body: [do: turn-right]] [utility 10] [id created-39]]
```

The learning plan 4 in the TUNNEL environment with 3 targets at a time also creates and accumulates plans by adjusting their utilities. Here, all plans existing at the time point 808 still exist until time point 2424. Their utility values are updated from time to time which means also that they are applied in different circumstances. However, at the last stage of the experiment their utilities drop. As can be seen at time point 2424, most plans that were retained from the beginning have low utility values so that at the end (time point 2929) they removed from the plan library. This indicates that learning plan 4 is not as effective as learning plan 1 in the TUNNEL environment although their overall performances are comparable.

Another interesting observation is that every plan created in the configuration 4-TUN-NEL-3 consists of a single step of action. Although some learnt plans do use a sequence structure (plan `created-10`, `created-21`, and `created-28`). However a plan with a sequence is not retained for a long time in the TUNNEL environment. It is shown in Fig. 36b that the number of plans learnt by the learning plan 3 is more fluctuating than the learning plan 1 and 4. The experiment of learning in the TUNNEL environment with 3 targets at a time has indicated that the learning plan that makes the rule-like structure of plan is more suitable than the other with a more complex action structure. This may explain how the strategy 1 and 4 are comparable in their performance.

As another example, the TURNINGS environment with 5 targets at a time shows that learning plan 4 still dominates the number of achievements as shown in Fig. 37a. In 3000 execution cycles, the agent can collect 55 targets using the learning plan 4. Plan 1, 2, and 3 can get 23, 17 and 15 targets respectively. On the other hand, the number of learnt plans over the 3000 execution cycles is still dominated by the learning plan 1, followed by learning plan 4. A distinct pattern can be seen in the number of learnt plans by the learning plan 4 along time points 1200 to 1800. In particular, the number of plans created decreases until around time point 1300 before it increases back to the maximum. This may be caused by a series of failures of the learnt plans or failures to find targets for some times.
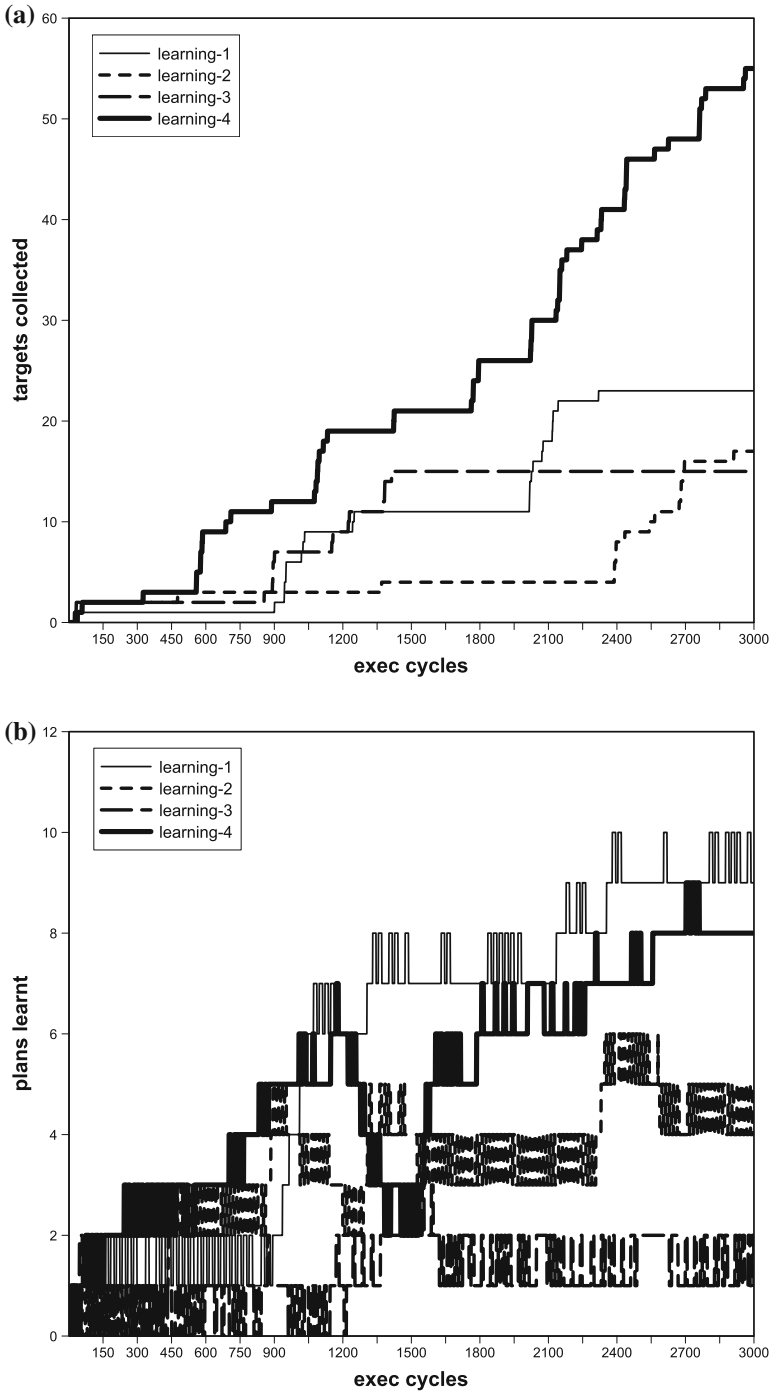
Fig. 37 Cumulative goals obtained and the number of plans learnt in the TURNINGS environment with target number 5. (**a**) Cumulative number of achievement. (**b**) Number of learnt plans

The trace of learnt plans from selected time points in the configuration 4-TURNINGS-5 is presented below:

Time point 808

```
[plan:
     [precond: [sense-buff [brick 1][brick 1][resource 1][resource 1][. 0]]]
     [id created-21] [body: [seq: [do: forward]]] [utility 9]]
[plan:
     [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][. 0]]]
     [body: [do: turn-right]] [utility 10] [id created-25]]
[plan:
     [precond: [sense-buff [brick 1][brick 1][. 0][. 0][. 0]]]
     [body [do: forward]] [id created-7] [utility 3]]
```

Time point 1111

```
[plan:
     [precond: [sense-buff [. 0][resource 1][. 0][brick 1][brick 1]]]
     [id created-2] [body [seq: [do: forward]]] [utility 9]]
[plan:
     [precond: [sense-buff [brick 1][brick 1][resource 1][resource 1][. 0]]]
     [id created-21] [body: [seq: [do: forward]]] [utility 9]]
[plan:
     [precond: [sense-buff [brick 1][brick 1][brick 1][brick 1][. 0]]]
     [id created-28] [body [seq: [do: forward] [do: turn-right]]] [utility 9]]
[plan:
     [precond: [sense-buff [brick 1][brick 1][. 0][. 0][. 0]]]
     [body [do: forward]] [id created-7] [utility 0]]
[plan:
     [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][. 0]]]
     [body [do: turn-right]] [id created-25] [utility 0]]
```

Time point 1414

```
[plan:
     [precond: [sense-buff [. 0][resource 1][. 0][brick 1][brick 1]]]
     [id created-2] [body [seq: [do: forward]]] [utility 9]]
[plan:
     [precond: [sense-buff [brick 1][brick 1][resource 1][resource 1][. 0]]]
     [id created-21] [body [seq: [do: forward]]] [utility 9]]
[plan:
     [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][. 0]]]
     [body [seq: [do: turn-right]]] [utility 10] [id created-40]]
```

Time point 1717

```
[plan:
     [precond: [sense-buff [. 0][resource 1][. 0][brick 1][brick 1]]]
     [id created-2] [body [seq: [do: forward]]] [utility 9]]
[plan:
     [precond:[sense-buff [brick 1][brick 1][resource 1][resource 1][. 0]]]
     [id created-21] [body: [seq: [do: forward-schema]]] [utility 9]]
[plan:
     [precond: [sense-buff [. 0][. 0][. 0][brick 1][brick 1]]]
     [utility 10] [id created-45] [body [repeat: [do: forward] [number 3]]]]
[plan:
     [precond: [sense-buff [brick 1][brick 1][brick 1][brick 1][. 0]]]
     [body [do: turn-right]] [utility 10] [id created-48]]
```

Time point 2424

```
[plan:
     [precond:[sense-buff [. 0][. 0][. 0][brick 1][brick 1]]]
     [id created-45] [body: [repeat: [do: forward]] [number 3]]
     [utility 8]]
[plan:
     [precond: [sense-buff [. 0][resource 1][. 0][brick 1][brick 1]]]
     [id created-2] [body [seq: [do: forward]]] [utility 8]]
```

```
[plan:
     [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][. 0]]]
     [body: [do: turn-right] [id created-55] [utility 7]]
[plan:
     [precond:[sense-buff [brick 1][brick 1][resource 1] [resource 1] [. 0]]]
     [id created-21] [body: [do forward]]] [utility 9]]
[plan:
     [precond: [sense-buff [brick 1][brick 1][. 0][. 0][. 0]]]]]]
     [body: [do: forward] [id created-65] [utility 7]]
[plan:
     [precond: [sense-buff [brick 1][brick 1][brick 1][brick 1][. 0]]]
     [body: [do: turn-right]]] [id-created-48] [utility 8]
```

Time point 2929

```
[plan:
     [precond: [sense-buff [. 0][. 0][. 0][brick 1][brick 1]]]
     [id created-45][body: [repeat: [do: forward]] [number 3]][utility 8]]
[plan:
     [precond: [sense-buff [. 0][resource 1][. 0][brick 1][brick 1]]]
     [id created-2] [body: [seq: [do: forward]]] [utility 8]]
[plan:
     [precond: [sense-buff [brick 1][brick 1][resource 1][resource 1][. 0]]]
     [id created-21] [body: [seq: forward-schema]] [utility 8]]
[plan:
     [precond: [sense-buff [brick 1][brick 1][. 0][resource 1][. 0]]]
     [id created-70] [body: [seq: [do: turn-right]]] [utility 9]]
[plan:
     [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][. 0]]]
     [body: [do: turn-right]] [id created-55] [utility 5]]
[plan:
     [precond: [sense-buff [brick 1][brick 1][brick 1][brick 1][. 0]]]
     [body: [do: turn-right]] [id created-48] [utility 2]]
[plan:
     [precond: [sense-buff [brick 1][brick 1][. 0][. 0][. 0]]]
     [body: [do: forward]] [id created-65] [utility 1]]
```

In the configuration 4-TURNINGS-3 there are more changes in the learnt plans over the period of time. The learning plan 4 now produces not just a single action plan but also plans involving complex actions. On time point 1717, the agent acquires a plan with a repetition structure (plan id created-45). This plan is retained until the end of the cycle. However, plans with the other kind of structure, like a sequence, mostly do not last long. This indicates that applying the repetition structure is more effective than the other actions structures in the TURNINGS environment.

The dominance of learning plan 4 followed by learning plan 1 in both previous presented configurations does not mean that learning plan 4 (or plan 1) is the best or optimum plan for this domain. Different configurations may show contrasting conditions. For example, Fig. 38a shows the number of targets achieved in the BARRIERS environment 3 targets at a time. In 3000 time cycles, the agent can collect 17 targets using the learning plan 2. As the runner up, the learning plan 4 obtained 11. Other learning plans achieve only few (3 for strategy 3 and 2 for strategy 1). In different situations, different actions structures may have different levels of effectiveness as learnt knowledge structure.

In Fig. 38b learning plan 2 and 4 acquire plans at about the same number. Although slighty lower than in the learning plan 2. Learning plan 1 and 3, on the other hand, are still struggling to acquire plans as their numbers are fluctuating over the period of time.

From the trace of the learnt plans, however, many plans are not retained and many are replaced in the learning plan 2. Although, the graph in Fig. 38b is showing that the learning plan 2 has a high number of plan, there are still many redundant plans acquired. The trace
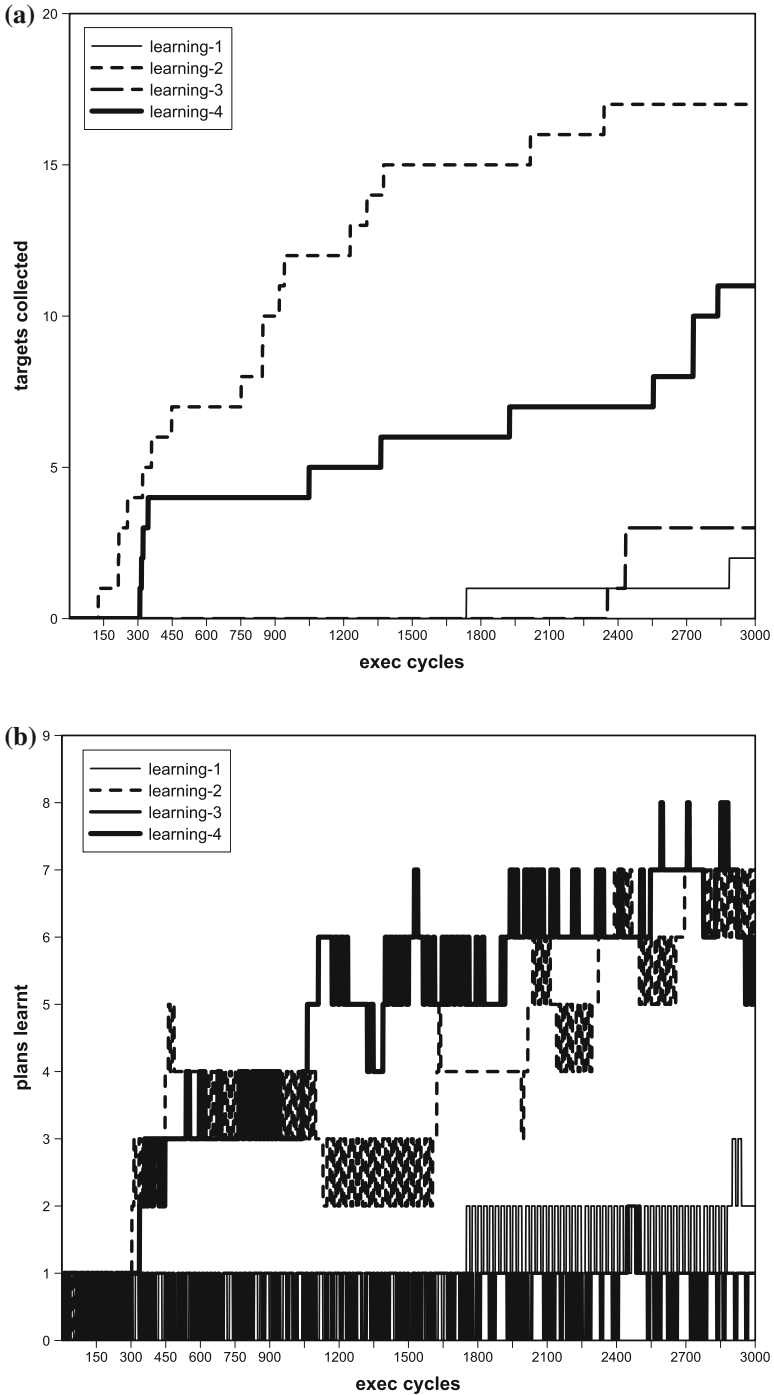
Fig. 38 Cumulative goals obtained and the number of plans learnt in the BARRIERS environment with target number 3. (**a**) Cumulative number of achievement. (**b**) Number of learnt plans

of plans in the configuration 2-BARRIERS-3, leaving out some redundant plans, are shown below.

Time point 303

```
[plan:
    [precond: [sense-buff [brick 1] [brick 1] [. 0] [. 0] [. 0]]]
    [utility 10] [id created-15]
    [body: [repeat: [do: forward] [number 2]]]]
```

Time point 505

```
[plan:
    [precond: [sense-buff [brick 1] [brick 1] [. 0] [. 0] [. 0]]]
    [utility 10] [id created-15]
    [body: [repeat: [do: forward] [number 2]]]]
[plan:
    [precond: [sense-buff [. 0] [. 0] [. 0] [brick 1] [brick 1]]]
    [utility 10] [id created-22]
    [body: [repeat: [do: forward] [number 5]]]]
```

Time point 2424

```
[plan:
    [precond: [sense-buff [brick 1] [brick 1] [. 0] [. 0] [. 0]]]
    [utility 10] [id created-96]
    [body: [repeat: [do: forward] [number 4]]]]
[plan:
    [precond: [sense-buff [. 0] [. 0] [brick 1] [. 0] [. 0]]]
    [body: [repeat: [do: forward] [number 1]]]
    [utility 10] [id created-101]]
```

Time point 2828

```
[plan:
    [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][brick 1]]]
    [body: [repeat: [do: turn-right] [number 1]]]
    [utility 10] [id created-116]]
```

The plans acquired by learning plan 2 are frequently changed and replaced. From time to time, the set of learnt plans available can be different. However, it is shown that some plans learnt at one time have the same preconditions with plans that are previously deleted. This means also the set of plans can be adapted for dealing with different situations.

In contrast, the learning plan 4 in the configuration 4-BARRIERS-3 can retain more knowledge than the learning plan 2 as presented below.

Time point 505

```
[plan:
    [precond: [sense-buff [. 0] [brick 1] [brick 1] [. 0] [. 0]]]
    [id created-13] [body: [seq: [do: turn-right]
    [do: forward] [do: turn-left]]] [utility 9]]
[plan:
    [precond: [sense-buff [. 0] [. 0] [. 0] [. 0] [. 0]]]
    [id created-14] [body: [seq: [do: turn-left]]] [utility 9]]
[plan:
    [precond: [sense-buff [brick 1] [. 0] [. 0] [. 0] [. 0]]]
    [body: [do: forward]] [id created-19] [utility 8]]
```

Time point 2727

```
[plan:
    [precond: [sense-buff [brick 1] [. 0] [. 0] [. 0] [. 0]]]
    [body: [do: forward]] [id created-19] [utility 7]]
[plan:
    [precond: [sense-buff [brick 1][brick 1][. 0][resource 1][. 0]]]
    [body: [do: turn-right]] [utility 10] [id created-59]]
```

```
[plan:
     [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][brick 1]]]
     [body: [do: turn-left]] [id created-46] [utility 7]]
[plan:
     [precond: [sense-buff [. 0][brick 1][brick 1][brick 1][. 0]]]
     [body: [do: turn-right]] [id created-74] [utility 0]]
[plan:
     [precond: [sense-buff [brick 1][brick 1][. 0][. 0][. 0]]]
     [body: [do: turn-right]] [id created-34] [utility 4]]
[plan:
     [precond: [sense-buff [. 0][brick 1][brick 1][. 0] . 0]]]
     [id created-13] [body: [seq: [do: turn-right]
     [do: forward] [do: turn-left]]] [utility 0]]
[plan:
     [precond: [sense-buff [. 0][. 0][. 0][. 0][. 0]]]
     [id created-14] [body: [seq: [do: turn-left]]] [utility 0]]
```

All plans learnt on the time point 505 are still retained in time point 2727, though their utilities mostly become 0, which means that sooner or later they will also be removed. It is also shown that in this configuration the plans learnt by the learning plan 4 consist of a mixture of plans with of single step actions and sequences.

Overall, the experiment with the vacuum explorer has demonstrated that the agent can learn using the intentional learning model to explore a simple two-dimensional grid environment. The experiment also indicated that different learning plans can produce different behavior in exploring the environment and obtaining the goals. The characteristics of the knowledge acquired from exploring the world are also shaped by how the learning plan is designed. The vacuum explorer experiment also reveals that, when *in-place* manipulation is applied with the manipulative abduction, the learnt knowledge may also be in flux. The learning and adaptation process may comprise both addition and removal of previously learnt knowledge. Although the results of the learning may not be optimal, it enables the learning to deal with changing situations in which some previously learnt knowledge are no longer consistent. In this case, the intentional learning process can be regarded as searching the right set of knowledge to achieve the goal while performing the task in the environment. However, the open challenge to evaluate learning in an environment like the vacuum explorer is to evaluate learning and plans acquisition when a small perturbation at the beginning leads to a big difference of performance at a later time. The differences are not just with respect to the agent's behavior, but also the knowledge acquired as the agent learns from its own experience.

4.3 Learning plans revisited

Interdependencies between learning strategies require consideration of the order in which a learning plan is activated and when a decision is about to be made.

In the rat's world domain, the learning plan that produces the optimal knowledge works sufficiently by itself. Each learning plan's performance is evaluated when it works individually. Although a complex structure of actions can be learnt, learning in the rat's world is still *monolithic* in which the same mechanism is employed in each learning episode. The main concern in the rat's world was limited to examining the capability of plan expression as the main representation used in learning. On the other hand, the vacuum explorer domain included multi-strategic learning in the evaluation (learning 4).

The results of experiments provide some hints on how to design a better learning plan. In a complex changing environment, the context of how and when learning is conducted is as important as the structure of the learnt plan. Moreover, one situation might need a different

focus of observation than another. The learning plan can be generalized to cover different contexts by adding more learning plans as higher level controllers of the learning processes.

A process of learning can be described as a meta-level plan that also functions as a commitment strategy. Based on its commitment to learn, the agent can maintain its intention to execute a plan to achieve a goal while actually testing a hypothesis of a new possible plan. The intention can be dropped when the agent believes that it no longer supports the hypothesis. The learning plan can be designed to deal with time constraints when the process of testing the hypothesis is made to focus only on a certain aspect of the domain problem. The structure of the hypothesis to be checked also determined the responsiveness of the agent towards changes in the environment. This can be possible as the meta-level plans representation can accommodate a wide range of plan expressions.

In the vacuum explorer domain, the learning plans are dynamically triggered at different times and situations instead of continuously executed, thanks to the structure of a learning plan that comprises leaning goal and precondition attributes. The triggering condition for most learning plans in that domain is simply the number of failures to achieve the goal. When the number exceeds a certain threshold the learning goal is posted to the goal structure and a learning process starts. Furthermore, the decision of which learning plan to select at one situation depends on the precondition attribute of that learning plan. For example, in the vacuum domain, the learning plan for updating the utility of another plan has a different precondition than another learning plan. The utility updating plan is only applied when a previously learnt plan just fails and requires improvement. On the other hand, other types of learning plan may only apply when no learnt plan can be found to achieve the goal and instead, the default plans are adopted, even though they are still selected at random as they have the same precondition.

Although the conditions applied as attributes of the learning plans in the current implementation are still limited to the agent's internal status of deliberation and plan adoption, it has been demonstrated that the learning can be designed to be triggered and employed only when necessary. It is also shown that parts of the learning involves recognition that a learning plan is applicable to a certain situation when the situation matches or satisfies the plan precondition.

Beyond the feature of dynamic activations of learning plans, there may still be other ways to improve learning plans for both the rat's world and vacuum explorer. In the rat's world domain, a hierarchical structure for learning plans may only be effective when the change in the environment is more complex than just swapping buttons' position. If the rule that determines how to obtain the reward can also change over a period of time, a composite structure of plans may be useful to learn. This learning plan that learns the composite structure may also be useful in dealing with different environment layouts so that learning can be general. In contrast with the implemented multi-strategic learning (learning 4) in the vacuum domain, which does not relate different structures of learnt plans from each other, the composite learning suggested here can create a plan with a mixed structure of actions.

It is also possible to extend the learning strategy so that the learning plan can produce a whole hierarchical structure of plans which consists of several plans and subplans. The learning plan acquires more than just a single plan. Some plans can be regarded as intermediate actions or subplans. The learning plan is not just constructing actions and preconditions but also setting up some landmarks. The landmarks can be used as milestones for subplans that can also be represented as subgoals. The goal attribute of each subplan can correspond to an observable landmark in the environment. The agent may try to find a condition marked as a subgoal before pursuing the rest of the plan to achieve the main goal.

In any case, learning processes and the structure of the environment are interdependent. On the one hand, the environment shapes the agent's knowledge during learning. On the other hand, the learning guides the agent to acquire the appropriate knowledge from the environment. Both sides must be considered when designing learning plans.

5 Reflections

The intentional learning model provides a way to describe learning processes as meta-level plans which can acquire complex structures of actions in a continuous dynamic situation. The capacity to learn the structure of plans on the run is beyond most conventional learning algorithms. Although the experiments conducted in this paper seem to suggest an ad-hoc setup, the learning plans used are derived from intuition about the structure of how the agent interacts with the environment, rather than on a formal analysis. In another setting, the learning plans can be specified based on heuristics, derived by an agent designer or a domain expert, that are also made to dynamically monitor and control the execution of particular intentions, rather than the rigid execution of an algorithm that processes a set of training data.

Although, under certain conditions, the use of a particular learning plan can make the agent learn effectively by quickly acquiring the appropriate knowledge, it does not mean that the intentional learning model is *better* than any conventional learning mechanism. Wolpert and Macready suggested the so called *No Free Lunch* (NFL) theorem which argues that for all possible objectives or performance measures, there is no single algorithm that can optimize better than any other [41]. In particular, all algorithms have exactly the same performance if they are averaged over all possible performance measures. The NFL theorem implies that it is impossible to find the most effective general purpose learning mechanism. It is only possible to find an effective learning strategy in a subset of situations of interest.

The NFL theorem supports the principle of intentional learning that to design effective learning, the characteristics of the environment and tasks of the agent must be considered. We identify several aspects that influence the effectiveness of the learning process as follows:

- *Observability of the environment* The capacity to observe and distinguish environment states influences the effectiveness of the learning. It is useful for the agent to have full observability (i.e. it can distinguish every state in the environment). However, sometimes too much observation can reduce the effectiveness of the learning as the chance of finding the right knowledge decreases.
- *Structure of the conjectured actions in the learning plan* When the learnt action structure matches with the rule of the environment, the optimal solution can be found instantly. However, an inappropriate structure of the learnt actions leads to a slower learning or in the worst case no significant improvement can be made.
- *Goals and triggers for learning* A learning goal determines when or in what condition a learning plan is adopted and executed. By putting the appropriate learning goal and triggering condition, a learning plan can be arranged to be invoked at the right time so that the performance of the agent can be improved much quicker.

Each individual aspect does not stand on its own. It is the combination of two or more aspects that may significantly change the outcomes.

Note that the above analysis is not meant to identify the best intentional learning plans, but rather to highlight key issues in the design of intentional learning agents. There is still an open issue in designing the appropriate learning mechanism in the intentional learning

model. Virtually any kind of process or behavior that involves intention monitoring and self-modifying plans can be expressed using this model. Consequently, the model also supports the realization of any kind of learning algorithm. The intentional learning model should not be seen as a rival to conventional learning algorithms but rather as a framework for supporting high-level learning strategies in the BDI agent architecture.

It is reasonable to consider that some prescribed heuristics for learning can only work effectively in a specific environment with only a narrow variation. The knowledge produced can be brittle and incapable of adapting to a major change in the environment. However, recent studies in heuristics have indicated that in some real world situations, simple heuristics for decision making outperform ones that are based on general laws of probability or logical coherence in respect of speed, simplicity, and accuracy [16]. Even though a type of heuristic by itself works only in a class of situations, a bundle of heuristics can make the agent adapt its behavior to different situations and changes. This approach is in line with our approach of building an intentional agent. A repository of plans must be provided with different classes of learning strategies so that the agent can adapt with different conditions.

One can view learning as an extensive search for appropriate knowledge that improves performance. Most learning algorithms can be perceived as a process of creating and testing hypotheses with experiences sampled on the run. Supported hypotheses are then used to adjust or re-configure knowledge. Knowledge adjustment can be done in many ways such as manipulating connections, statistical measures, inductions, or analytical inferences.

In most conventional learning algorithms, each learning period is processed by the same learning procedure regardless of the computational burden that might be imposed. The end product of the learning would be a model reflecting the environment with a probabilistic distribution, a utility function, or some consistent propositions for algorithms based on logic. This learning process is conducted by a separate module or instance of a program that tries to improve domain level performance. Consequently, most conventional learning algorithms are assumed to get their experiences from external sources. Reinforcement learning algorithms, that are commonly used for learning autonomous actions, mainly rely on external sources like observation and rewards [1,28,39]. They still neglect the capacity of the agent to have intentions and to use recipes with structured actions.

In contrast, the novelty of the intentional learning model is that it puts the learning process as a commitment of the agent. In more detail, a decision to commit to a certain intention can be based on the capacity to validate the hypothetical structure of knowledge. Consequently, the learning process uses the actual state of the intention as the source of experience. A domain expert has more freedom to supply the agent with a set of rules-of-thumb to handle commitment for learning, thanks to the expressive power of the plan representation.

6 Conclusion

This paper presented a model of intentional learning based on the BDI agent architecture. The model provides a mechanism that uses meta-level plans (so-called learning plans) to conduct plan acquisition while controlling deliberation and execution. Procedural knowledge is acquired by a manipulative abduction process in which the meta-level plan makes a hypothesis and tests it on the run. Some examples and patterns of such learning plans that can be used in this intentional learning model have been presented. It was shown that patterns of learning plans can be made to acquire different structure of actions, including reactive plan structures and complex actions.

The intentional learning model can be regarded as a framework for developing learning agents. It allows the agent designer to explicitly describe how the agent can learn from its own experiences as an integral part of its domain level behavior. The model provides a notation and a set of primitive operators to describe meta-level plans so that it is possible to monitor the status of intentions and modify plans on the run. Beyond conventional learning approaches to learning, it is possible to describe learning plans that can acquire plans with complex structure and hierarchical relationships of intentions and plans, as the agent interacts with the environment.

It is suggested in this paper that knowledge of the structure of tasks and the characteristics of the environment can be the basis for the agent designer to develop learning plans. Adjustments to the learning plans can be made by manipulating their preconditions, structures of the plan bodies, and the goals of the target plans. It is also possible to incorporate heuristics, which are derived from the structure of environment, as templates for building learning plans. However it is still up to the designer's creativity and domain knowledge to identify and apply the appropriate set of learning plans for a given problem. A major future direction of this research involves identifying appropriate *design patterns* for guiding the construction of intentional learning agents based on problem descriptions.

References

1. Alpaydin, E. (2004). *Introduction to machine learning*. Cambridge: MIT Press.
2. Bauer, B., & Odell, J. (2005). UML 2.0 and agents: How to build agent-based systems with the new uml standard. In *Engineering applications of artificial intelligence* (Vol. 18, pp. 141–157). Elsevier.
3. Bereiter, C., & Scardamalia, M. (1989). Intentional learning as a goal of learning. In L. B. Resnick (Ed.), *Knowing, learning, and instruction: Essays in honor of Robert Glaser* (Chap. 12, pp. 361–392). Hillsdale: Erlbaum associates.
4. Bordini, R. H., Hubner, J. F., & Wooldridge, M. J. (2007). *Programming multi-agent systems in Agent-Speak using Jason*. John Wiley & Sons.
5. Bratman, M. E. (1987). *Intention, plans and practical reason*. Cambridge: Harvard University Press.
6. Bratman, M. E. (1990). What is intention? In P. R. Cohen, J. Morgan, & M. E. Pollack (Eds.), *Intentions in communication* (Chap. 2, pp. 15–31). Cambridge: MIT Press.
7. Bratman, M., Israel, D., & Pollack, M. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence, 4*, 349–355.
8. Braubach, L., & Pokahr, A. Jadex BDI agent system. http://vsis-www.informatik.uni-hamburg.de/projects/jadex/.
9. da Silva, V. T., Noya, R. C., & de Lucena, C. J. P. (2005). Using the UML 2.0 activity diagram to model agent plans and actions. In *AAMAS '05: Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems* (pp. 594–600). New York, ACM.
10. Dastani, M., van Riemsdijk, M. B., & Meyer, J.-J. C. (2005). Programming multi-agent systems in 3APL. In R. H. Bordini, M. Dastani, J. Dix, & A. E.-F. Seghrouchni (Eds.), *Multi-agent programming: Languages, platforms and applications* (Chap. 2, pp. 39–67). New York: Springer.
11. Ferber, J. (1999). *Multi-agent systems: An introduction to distributed artificial intelligence*. Harlow: Addison-Wesley.
12. Fowler, M. (2003). *UML distilled 3rd edition: A brief guide to the standard object modeling language* (3rd ed.). Addison-Wesley.
13. Georgeff, M. P., & Lansky, A. L. (1987). Reactive reasoning and planning. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-87)* (pp. 677–682). Seattle.
14. Georgeff, M., Pell, B., Pollack, M., Tambe, M., & Wooldridge, M. (1999). The belief-desire-intention model of agency. In J. Müller, M. P. Singh, & A. S. Rao (Eds.), *Proceedings of the 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures, and Languages (ATAL-98)*, Vol. 1555 of *Lecture Notes in Artificial Intelligence* (pp. 1–10). Heidelberg: Springer-Verlag.
15. Gigerenzer, G., & Selten, R. (Eds.) (2001). *Bounded rationality: The adaptive toolbox*. Cambridge: MIT Press.

16. Gigerenzer, G., Todd, P. M., & The ABC Research Group. (1999). *Simple heuristics that make us smart*. New York: Oxford University Press.
17. Haddadi, A., & Sundermeyer, K. (1996). Belief-desire-intention agent architectures. In G. M. P. O'Hare & N. R. Jennings (Eds.), *Foundations of distributed artificial intelligence* (pp. 169–185). New York: Wiley.
18. Howden, N., Ronnquist, R., Hodgson, A., & Lucas, A. (2001). JACK intelligent agents-summary of an agent infrastructure. In T. Wagner & O. F. Rana (Eds.), *Proceedings of the 5th International Conference on Autonomous Agents, Workshop on Infrastructure for Agents, MAS and Scalable MAS* (pp. 251–257). New York: ACM Press.
19. Huber, M. J. (1999). JAM: A BDI-theoretic mobile agent architecture. In *Proceedings of the Third International Conference on Autonomous Agents* (pp. 236–243). New York: ACM Press.
20. Ingrand, F. F., & Georgeff, M. P. (1990). Managing deliberation and reasoning in real-time AI systems. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning*, San Diego.
21. Ingrand, F., Georgeff, M. P., & Rao, A. S. (1992). An architecture for real-time reasoning and system control. *IEEE Expert: Intelligent Systems and Their Applications, 7*(6), 34–44.
22. Kakas, A. C., Kowalski, R. A., & Toni, F. (1993). Abductive logic programming. *Journal of Logic and Computation, 2*, 719–770.
23. Kinny, D., & Georgeff, M. P. (1991). Commitment and effectiveness of situated agents. In *Proceedings of the 12th International Joint Conference of Artifical Intelligence 1991* (pp. 82–88). Morgan Kaufmann.
24. Kinny, D., Georgeff, M. P., & Rao, A. S. (1996). A methodology and modelling technique for systems of BDI agents. In W. Van de Velde & J. W. Perram (Eds.), *Agents Breaking Away, Proceedings of 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Vol. 1038 of *Lecture Notes in Computer Science* (pp. 56–71). Springer.
25. Kukla, A., & Walmsley, J. (2006). *Mind: A historical and philosophical introduction to the major theories*. Indianapolis: Hackett Publishing.
26. Magnani, L. (2001). *Abduction, reason, and science: Processes of discovery and explanation*. New York: Kluwer Academic/Plenum Publishers.
27. Magnani, L. (2007). Mimetic minds: Meaning formation through epistemic mediators and external representations. In A. Loula, R. Gudwin, & J. Quieroz (Eds.), *Artificial cognition systems* (pp. 327–357). Idea Group.
28. Mitchell, T. (1997). *Machine learning*. Cambridge: McGraw-Hill.
29. Morley, D., & Myers, K. (2004). The SPARK agent framework. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-04)* (pp. 712–719). Washington, IEEE Computer Society.
30. Ram, A., & Leake, D. B. (Eds.). (1995). *Goal-driven learning*. Cambridge: MIT Press.
31. Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In R. van Hoe (Ed.), *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. The Netherlands: Eindhoven.
32. Rao, A. S., & Georgeff, M. P. (1995). BDI agents: From theory to practice. In V. R. Lesser & L. Gasser (Eds.), *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95)* (pp. 312–319). Cambridge: MIT Press.
33. Russell, S., & Norvig, P. (2003). *Artificial intelligence: A modern approach* (2nd ed.). London: Prentice Hall.
34. Schut, M., Wooldridge, M., & Parsons, S. (2004). The theory and practice of intention reconsideration. *Journal of Experimental and Theoretical Artificial Intelligence, 16*, 261–293.
35. Shanahan, M. (1989). Prediction is deduction but explanation is abduction. In N. S. Sridharan (Ed.), *Proceedings of the 11th International Joint Conference on Artificial Intelligence* (pp. 1055–1060). Morgan Kauffman.
36. Shanahan, M. (2000). An abductive event calculus planner. *Journal of Logic Programming, 44*, 207–240.
37. Simon, H. A. (1982). *Models of bounded rationality*. Cambridge: MIT Press.
38. Subagdja, B. (2007). *Intentional learning in bounded-rational agents*. PhD thesis, Department of Information Systems, University of Melbourne, 2007.
39. Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge: MIT Press.
40. Tisue, S., & Wilensky, U. (2004). Netlogo: Design and implementation of a multi-agent modeling environment. In *Proceedings of Agent 2004 Conference on Social Dynamics: Interaction, Reflexivity and Emergence*, Chicago.
41. Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorem for optimization. *IEEE Transaction on Evolutionary Computation, 1*(1), 67–82.
42. Wooldridge, M. J. (2000). *Reasoning about rational agents*. Cambridge: MIT Press.
43. Wooldridge, M. J. (2001). *An introduction to multiagent systems*. Chichester: Wiley.