# Learning Context Conditions for BDI Plan Selection

Dhirendra Singh, Sebastian Sardina, and Lin Padgham
RMIT University
Melbourne, Australia
{dhirendra.singh,sebastian.sardina,lin.padgham}@rmit.edu.au

Stéphane Airiau
University of Amsterdam
Amsterdam, The Netherlands
s.airiau@uva.nl

## ABSTRACT

An important drawback to the popular Belief, Desire, and Intentions (BDI) paradigm is that such systems include no element of learning from experience. In particular, the so-called *context conditions* of plans, on which the whole model relies for plan selection, are restricted to be boolean formulas that are to be specified at design/implementation time. To address these limitations, we propose a novel BDI programming framework that, by suitably modeling context conditions as *decision trees*, allows agents to *learn* the probability of success for plans based on previous execution experiences. By using a probabilistic plan selection function, the agents can balance exploration and exploitation of their plans. We develop and empirically investigate two extreme approaches to learning the new context conditions and show that both can be advantageous in certain situations. Finally, we propose a generalization of the probabilistic plan selection function that yields a middle-ground between the two extreme approaches, and which we thus argue is the most flexible and simple approach.

## Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Intelligent agents; I.2.6 [**Learning**]: Induction

## General Terms

Algorithms, Performance

## Keywords

BDI, Learning (single and multi-agent)

## 1. INTRODUCTION

In this paper, we are concerned with one of the key aspects of the BDI agent-oriented programming paradigm, namely, that of *intelligent plan selection* [7, 12]. Specifically, we explore the details of how effective plan selection can be learnt based on ongoing experience.

There are a plethora of agent programming languages and development platforms in the BDI tradition, such as PRS [7], JACK [5], 3APL [9] and 2APL [6], Jason [3], and SRI's SPARK [11], among others. Generally speaking, these systems enable *abstract plans* written by programmers to be combined and used in real-time, in a way that is both flexible and robust. Concretely, a BDI agent is built around a *plan library*, a collection of pre-defined *hierarchical plans* indexed by goals and representing the standard operational procedures of the domain (e.g., landing a plane). The so-called *context condition* attached to each plan states the conditions under which the plan is a sensible strategy to address the corresponding goal in a given situation (e.g., it is not raining). The execution of a BDI system relies then entirely on *context sensitive subgoal expansion*, allowing agents to "act as they go" by making *plan choices* at each level of abstraction with respect to the current situation.

The fact that both the actual behaviours (the plans) and the situations for which they are appropriate (their context conditions) are fixed at design time has important implications for the whole programming approach. First, it is often difficult or impossible for the programmer to craft the *exact* conditions under which a plan would succeed. Second, once deployed, the plan selection mechanism is fixed and may not adapt to potential variations of different environments. Finally, since plan execution often involves interaction with a *partially observable* external world, it is desirable to measure success in terms of probabilities rather than boolean values.

The authors have been exploring the nuances in learning within the hierarchical structure of a BDI program [1] where it can be problematic to assume a mistake at a higher level in the hierarchy, when a poor outcome may have been related to a mistake in selection further down (Section 3). In this paper we show how the scheme which does not take account of this fact, can at times lead to a complete inability to learn (Section 5). We outline two approaches which we have described previously: a conservative approach which takes account of the structure, considering failures only when decisions made during the execution are deemed sufficiently "informed.", and an aggressive approach which ignores the structure and had initially seemed preferable (Section 4). We then describe a new approach which instead of being conservative in which training examples are used, includes a *confidence* measure based on how much the agent has explored the space of possible executions of a given plan (Section 6). The more this space has been "covered" by previous executions, the more the agent "trusts" the estimation of success provided by the plan's decision tree. This approach to selection, when combined with the aggressive approach to training examples achieves a flexible and simple compromise between the previous two approaches.

Our approach is fully compatible with the usual methodology to plan selection using programmed formula based context conditions. In real applications we would in fact expect the learning to "refine" initially provided selection conditions. For simplicity, however, context conditions are learnt from scratch in our experimental work.

**Cite as:** Learning Context Conditions for BDI Plan Selection, Dhirendra Singh, Sebastian Sardina, Lin Padgham, and Stéphane Airiau, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. XXX-XXX.

## 2. BDI PROGRAMMING

BDI agent-oriented programming is a popular, well-studied, and practical paradigm for building intelligent agents situated in complex and dynamic environments with (soft) real-time reasoning and control requirements [2, 7]. A BDI-style agent system consists, basically, of a *belief base* (the agent's knowledge about the world), a set of recorded pending *events* or *goals*, a plan library (the typical operational procedures of the domain), and an intention base (the plans that the agent has already committed to and is executing).

The basic reactive goal-oriented behavior of BDI systems involves the system responding to events—the inputs to the system—by committing to handle one pending event-goal, *selecting a plan from the library*, and placing its program into the intention base. A *plan* in the plan library is a rule of the form $e : \psi \leftarrow \delta$: program $\delta$ is a reasonable strategy to resolve event-goal $e$ whenever the context condition $\psi$ is believed true. Among other operations, program $\delta$ typically includes the execution of *primitive actions* ($act$) in the environment and the "posting" of new *subgoal events* ($!e$) that ought to be resolved by selecting (other) suitable plans. A plan may be selected for addressing an event if it is *relevant* and *applicable*, that is, if it is a plan designed for the event in question and its context condition is believed true, respectively. In contrast with traditional planning, execution happens at each step. The assumption is that the use of plans' context-preconditions to make choices as late as possible, together with the built-in goal-failure mechanisms, ensures that a successful execution will eventually be obtained while the system is sufficiently responsive to changes in the environment.

For the purposes of this paper, we shall mostly focus on the plan library. It is not hard to see that, by grouping together plans responding to the same event type, the plan library can be seen as a set of *goal-plan tree* templates: a goal (or event) node has children representing the alternative relevant plans for achieving it; and a plan node, in turn, has children nodes representing the subgoals (including primitive actions) of the plan. These structures, can be seen as AND/OR trees: for a plan to succeed all the subgoals and actions of the plan must be successful (AND); for a subgoal to succeed one of the plans to achieve it must succeed (OR).

Consider, for instance, the goal-plan tree structure depicted in Figure 1. A link from a goal to a plan means that this plan is relevant (i.e., potentially suitable) for achieving the goal (e.g., $P_1 \ldots P4$ are the relevant plans for event-goal $G$); whereas a link from a plan to a goal means that the plan needs to achieve that goal as part of its (sequential) execution (e.g., plan $P_A$ needs to achieve goal $G_{A1}$ first and then $G_{A2}$). For compactness, an edge with a label $\times n$ states that there are $n$ edges of such type. Leaf plans directly interact with the environment and so, in a given world state, they can either succeed or fail when executed; this is marked accordingly in the figure *for some particular world* (of course such plans may behave differently in other states). In some world, given successful completion of $G_A$ first, the agent may achieve goal $G_B$ by selecting and executing $P_B$, followed by selecting and executing 2 leaf working plans to resolve goals $G_{B1}$ and $G_{B2}$. If the agent succeeds with goals $G_{B1}$ and $G_{B2}$, then it succeeds for plan $P_B$, achieving thus goal $G_B$ and the top-level goal $G$ itself. There is no possible successful execution, though, if the agent decides to carry on any of the three plans labelled $P'_{B2}$ for achieving low-level goal $G_{B2}$.

Clearly, the problem of *plan-selection* is at the core of the BDI approach: *which plan should the agent commit to in order to achieve a certain goal?* This problem amounts, at least partly, to what has been referred to as *means-end analysis* in the agent foundational literature [12, 4], i.e., the decision of *how* goals are achieved. To tackle the plan-selection task, state-of-the-art BDI systems leverage domain expertise by means of the context conditions of plans. However, crafting fully correct context conditions at design-time can be a demanding and error-prone task. Also, fixed context conditions do not allow agents to adapt to changing environments. Below, we shall provide an extended BDI framework that allows agents to learn/adapt plans' context conditions, and discuss and empirically evaluate different approaches for such learning task.

## 3. A BDI LEARNING FRAMEWORK

The problem that we are interested in is as follows: *given past execution data and the current world state, determine which plan to execute next in order to address an event*.

To address this "learnable" plan-selection task, we start by modeling the context condition of plans with *decision trees*, rather than with logical formulas.[1] Decision trees [10] provide a natural classification mechanism for our purposes, as they can deal well with noise (generally due to partially observable and predictable environments), and they are able to support disjunctive hypotheses. They are also readily convertible to rules, which are the standard representation for context conditions in BDI systems.

We associate each plan in the agent's library with a decision tree that classifies world states into an expectation of whether the plan will succeed or fail. Then for each relevant plan, its decision tree (induced based on previous executions) gives the agent information regarding how likely it is to succeed/fail in a particular world state.

Given this new context for BDI programming, there are two issues that ought to be addressed. First, one has to decide *when and what kind of execution data the agent should collect in order to be able to "learn" (useful) decision trees for plans*. Roughly speaking, data is collected regarding whether a plan is considered to have succeeded or failed in the world for which it was selected. Whereas successful executions are always recorded, the recording of failure runs of a plan may be subject to some analysis; this is the topic of the following section.

The second issue to be addressed is how to use the plans' decision trees for plan selection. More concretely: *given a goal to be resolved and a set of relevant plans with their corresponding context decision trees, what plan should the agent commit for execution?* Typical BDI platforms offer various mechanisms for plan selection, including plan precedence and meta-level reasoning. However, these mechanisms are pre-programmed and do not take into account the experience of the agent. In our framework for context learning, we must consider the standard dilemma of *exploitation* vs *exploration*. To that end, we use an approach in which plans are selected with a probability proportional to their relative expected success (in the world state of interest). Later, in Section 6, we discuss how to further enhance such plan selection by considering how much each candidate plan has been explored relative to its "complexity."

For the purpose of our analysis, we have used algorithm J48, a version of c4.5 [10], from the well-known weka learning package [17]. Currently we recreate decision trees from scratch after each new outcome is recorded. Of course, for a real-world implementation, one should appeal to algorithms for *incremental* induction of the decision tree, such as those described in [14, 16].

The weka J48 algorithm for inducing decision trees aims to balance compactness of representation with accuracy. Consequently, it maintains in each decision tree information about the number of instances (or world states in our case) from the training data correctly and incorrectly classified by each decision tree leaf node. Subsequently, whenever a plan's decision tree is used to classify a

[1]The logical formulae of the classical BDI framework can of course be combined with decision trees.

new instance (world state), weka returns not only the classification (i.e. success or failure), but also a classification probability (i.e. to what degree it believes that the classification is correct). We then use this probability as an estimate of the plan's chances of success for the world in question.

Finally, we should point out a number of assumptions that were made in order to focus on the core issues we are concerned with. First, we assume that actions in the environment may fail with some probability (if an action is not possible in a world state this probability is 1.0). This is a simple way to capture non-deterministic failures caused either by imperfect execution or external changes in the environment. A success on the other hand is always attributed only to the agent's actions. Second, we consider the execution of a single intention; learning in the context of multiple, possibly interacting, intentions poses other extra challenges that would complicate our study substantially (see [15]). Lastly, we assume no automated failure handling, whereby the BDI system retries a goal with alternative options if the selected plan happens to fail. Integrating failure handling would complicate our implementation framework and the understanding of the basic mechanisms. For instance, if an alternative plan were to succeed after the initial failure then care must be taken in propagating this outcome to the parent as the success may have been caused precisely because the first choice failed in a way that enabled the second one to succeed.

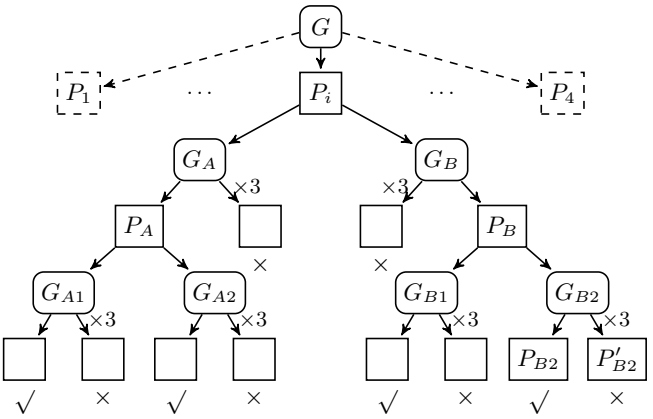



**Figure 1: Goal-plan hierarchy $\mathcal{T}_3$. There are $2^4$ worlds whose solutions are distributed evenly in each of the $4$ top level plans. Successful execution traces are of length $4$. Within each subtree $P_i$, BUL is expected to perform better for a given world, but it suffers in the number of worlds. Overall, ACL and BUL perform equally well in this structure.**

# 4. CONTEXT LEARNING: 2 APPROACHES

With the classical BDI programming framework extended with decision trees and a probabilistic plan selection scheme, we are now ready to develop mechanisms for learning context decision trees based on online experiences, in order to improve plan selection over time. To that end, in this section, we explore two approaches for learning the context condition of plans.

Recall that our objective is to learn which plans are best for achieving a particular goal in the various world states that may ensue. Given that, in this work, we have no measure of cost for plans,[2] a good plan for a given world state is simply one which (usually) succeeds in such state. In order to learn the context decision tree for a plan, the agent keeps track of previous experiences it has had when running the plan in question. More precisely, if a plan $P$ is tried in world state $w$ with certain outcome $o \in \{succ, fail\}$, the agent may record the tuple $\langle w, o\rangle$ as part of the training set for plan $P$. Interestingly, while it is always meaningful to record successful executions, some failures may not be worth recording. Based on this observation, we shall develop and compare two different algorithms that differ on how past experiences are taken into account by the agent. Before then, though, let us explain better this issue by means of an example.

Consider the example in Figure 1. Suppose that in some execution, plan $P_i$, for some $i \in \{1, \ldots, 4\}$, is selected in order to resolve top-goal $G$ in some world state $w_1$. The plan involves, in turn, the successful resolution of sequential goals $G_A$ and $G_B$. Suppose further that subgoal $G_A$ has been resolved successfully, yielding new state $w_2$, and that plan $P_B$ has been chosen next to try and achieve the second subgoal $G_B$. Suppose next that the first subgoal of plan $P_B$, namely $G_{B1}$ has been successfully resolved, yielding new state $w_3$, but that the non-working plan $P'_{B2}$ for subgoal $G_{B2}$ is selected in $w_3$ and execution thus *fails*. As there is no failure recovery, this failure will be propagated upwards in the hierarchy, causing goals $G_{B2}$ as well as $G_B$ and top-level goal $G$ itself to fail. First of all, the failure (in world state $w_3$) must be recorded in the decision tree of the plan where the failure originated, namely, plan $P'_{B2}$. Such bottom-level plans have no subgoals, so they interact with the external world directly, and over time we can expect to learn such interactions. On the other hand it is unclear, as we will show below, whether failure should also be recorded in the decision trees for plans higher up in the hierarchy (i.e. plans $P_B$ and $P_i$).

In order to discuss further *which* data should be recorded *where*, we define the notion of an *active execution trace*, as a sequence of the form $G_0[P_0 : w_0] \cdot G_1[P_1 : w_1] \cdot \ldots \cdot G_n[P_n : w_n]$, which represents the sequence of currently active goals, along with the plans which have been selected to achieve each of them, and the world state in which the selection was made—plan $P_i$ has been selected in world state $w_i$ in order to achieve the $i$-th active subgoal $G_i$. In our example, the trace at the moment of failure is as follows:

$$\lambda = G[P_i : w_1] \cdot G_B[P_B : w_2] \cdot G_{B2}[P'_{B2} : w_3].$$

So, when the final goal in $\lambda$ fails, namely $G_{B2}$, it is clear that the decision tree of the plan being used to achieve this goal ought to be updated, and a failure should be recorded for the world state $w_3$ against the decision tree attached to plan $P'_{B2}$. By recording every outcome for the lowest plans, i.e., plans with no subgoals, the system eventually learns how such plans perform in the environment.

What is more difficult to determine is whether the decision trees of plans associated with *earlier goals* in $\lambda$ should also be updated. More concretely, should failure cases in world states $w_2$ and $w_1$ be recorded against plans $P_B$ and $P_i$, respectively? The point is that it is conceivable that the failure of subgoal $G_{B2}$ in plan $P_B$, for instance, could indeed have been avoided, had the alternative plan $P_{B2}$, been chosen instead. Therefore, recording a failure against plan $P_B$ would not be justified—it is not true that plan $P_B$ is a "bad" choice in world state $w_2$. Informally, one could argue that it is more appropiate to *wait* before recording failures against a plan until one is reasonably confident that subsequent choices down the goal-plan tree hierarchy were "well informed." In our case, if the agent knows that the plan selection for goal $G_{B2}$ was as good and informed as possible, then recording the failure for world $w_2$ in plan $P_B$ would also be justified. Similarly, if the agent considers that the plan selection for subgoal $G_B$ was an informed choice,

[2]This could also be a useful addition, but is not part of standard BDI programming languages.

then recording the failure for world $w_1$ against $P_i$'s decision tree would be justified too.

The judgement as to whether plan choices were sufficiently "well informed," is however not a trivial one. A failed plan $P$ is considered to be *stable* for a particular world state $w$ if the rate of success of $P$ in $w$ is changing below a certain threshold $\epsilon$. In such a case, the agent can start to build confidence about the applicability level of $P$. The stability notion extends to goals as follows: a failed goal is considered *stable* for world state $w$ if all its relevant plans are stable for $w$. When a goal is stable, we regard the plan selection for such goal as a "well informed" one. Thus, a failure is recorded in the plan for a given world if the subgoal that failed is stable for the respective world in which it was resolved. In our example, we record the failure in plan $P_B$ ($P_i$) if goal $G_{B2}$ ($G_B$) is deemed stable in world state $w_3$ ($w_2$), that is, if the selection of option $P'_{B2}$ ($P_B$) was an informed one.

The RecordFailedTrace algorithm below shows how a failed execution run $\lambda$ is recorded. Function StableGoal$(G, w, k, \epsilon)$ returns true iff goal $G$ is considered *stable* for world state $w$, for success rate change threshold $0 < \epsilon \leq 1$ and minimal number of executions $k \geq 0$. The algorithm starts by recording the failure against the last plan $P_n$ in the trace. Next, if the choice of executing plan $P_n$ to achieve goal $G_n$ was deemed an informed one (that is, goal $G_n$ was stable for $w_n$), then the procedure should be repeated for the previous goal-plan nodes, if any. If, on the other hand, the last goal $G_n$ in the trace is not considered stable enough, the procedure terminates and no more failure data is assimilated. Observe that, in order to update the decision tree of a certain plan that was chosen along the execution, it has to be the case that the (failed) decisions taken during execution must have all been informed ones. Note that the stability idea only applies to failures since successes are always recorded.

**Algorithm 1** RecordFailedTrace$(\lambda, k, \epsilon)$

**Require:** $\lambda = G_0[P_0 : w_0] \cdot \ldots \cdot G_n[P_n : w_n]$; $k \geq 0$; $\epsilon > 0$
**Ensure:** Propagates DT updates for plans
1: RecordWorldDT$(P_n, w_n, \mathit{fail})$
2: **if** StableGoal$(G_n, w_n, k, \epsilon) \wedge |\lambda| > 1$ **then**
3: $\quad \lambda' := G_0[P_0 : w_0] \cdot \ldots \cdot G_{n-1}[P_{n-1} : w_{n-1}]$
4: $\quad$ RecordFailedTrace$(\lambda', k, \epsilon)$ `// recursive call`
5: **end if**

So, in the remainder of the paper, we shall consider two learning approaches compatible with the framework developed in the previous section. The first, which we call *aggressive concurrent learning* (ACL), corresponds to the more traditional approach where all data is always assimilated by the learner, that is, we take $\epsilon = 1$ and $k = 0$. In other words, every plan and every goal is always considered stable and, as a result, a failure in a plan is always recorded. The assumption is that misleading information, as discussed above, will eventually disappear as noise. The second one, which we refer to as *bottom-up learning* (BUL), is more cautious and records a failure execution experience when the success rate has stabilised i.e. is not changing by more than $\epsilon$. In our work, we have taken $\epsilon = 0.3$ and $k = 3$, that is, the context condition of a plan is considered stable (for a world state) if at least 3 past execution experiences have been recorded and the change in the rate of success over the last two experiences is less than 0.3. Note that the lower $\epsilon$ is and the higher $k$ is, the more conservative the agent is in considering its decisions "well informed."

In the following section, we shall explore these two approaches against different programs with different structures.

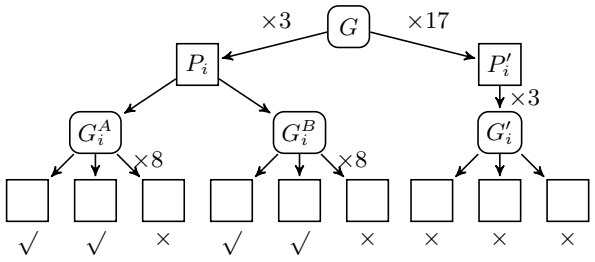



**Figure 2: Goal-plan tree structure $\mathcal{T}_1$. To succeed, an agent needs to make three correct choices, including selecting $P_i$ at the top-level. The solutions to $2^3$ worlds are distributed evenly in the 3 plans $P_i$. ACL outperforms BUL in this structure.**

# 5. EXPERIMENTAL RESULTS

In order to explore the difference between BUL and ACL, we set up testbed programs composed of several goals and plans combined in a hierarchical manner and yielding goal-plan tree structures of different shapes.[3] In particular, we crafted goal-plan tree structures representing different cases of BDI programs with one main top-level goal to be resolved. In addition, for each structure there is always a way of addressing the main goal, i.e. there is at least one successful execution of the top-level goal provided the right plan choices are made. Observe that such successful (plan) choices are different for different world states. When it comes to describing the possible (observable) world states, we have used a set of logical (binary) propositions, representing the so-called fluents or features of the environment that are observable to the agent (e.g., fluent proposition $DoorLocked$ states whether the door is believed open or not). Finally, we assume the agent is acting in a non-deterministic environment in which actions that are expected to succeed may still fail with some probability. In our experiments we assign a 0.1 probability of unaccounted failure to all actions.[4]

The experiments consisted in posting the top-level goal repetitively under random world states, running the corresponding BDI learning agent, and finally recording whether the execution terminated successfully or not. We calculate the average rate of success of the goal by first averaging the results at each time step over 5 runs of the same experiment, and then smoothing using a moving average of the previous 100 time steps to get the trends reported in the figures. We ran the tests with both a BUL-based agent and a ACL-based agent, ensuring the same sampling of random world states for each.

From our set of experiments, we have selected three hierarchical structures that best illustrate the results that we have obtained, namely:

**(Tree $\mathcal{T}_1$; Figure 2)** For each world state, the goal has a few plans that can succeed (plans $P_i$), but many other options of comparable complexity that are bound to fail (plans $P'_i$).[5] Under this type of structure, an ACL-based agent will generally per-

[3]We have implemented the learning agent system in the JACK BDI platform [5]. The fact that JACK is Java based and provides powerful meta-level reasoning capabilities allows us to integrate weka and probabilistic plan-selection mechanisms with little effort. Nonetheless, all the results are independent of this choice and could be reproduced in other BDI implementations.

[4]See Discussion section on how our results generalize to a framework with world state built from non-binary fluents and with more complex accounts for stochastic actions.

[5]Here, plan complexity refers to the size of the fully expanded plan,

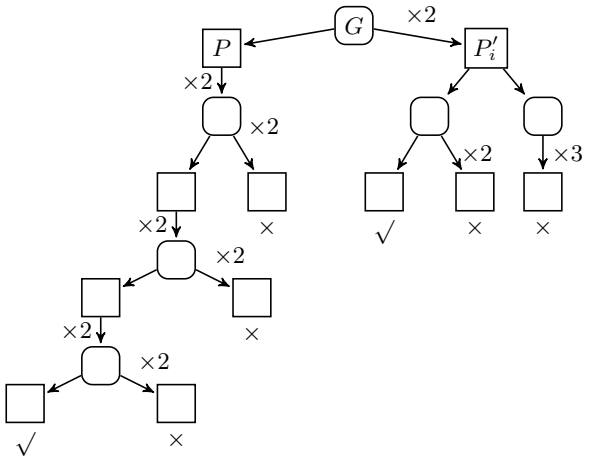



**Figure 3: Goal-plan tree hierarchical structure $\mathcal{T}_2$. Successful execution requires numerous correct choices including $8$ correct action nodes. The solutions to $2^3$ worlds are in the plan labelled $P$. BUL outperforms ACL in this structure.**

form better than an agent using the learning BUL approach.

**(Tree $\mathcal{T}_2$; Figure 3)** For each world state, the goal has one plan that can succeed (plan $P$), and a few others that would fail. However, the plan containing the solution is of substantially higher-complexity. In this structure, a BUL-based agent will outperform an ACL-based one.

**(Tree $\mathcal{T}_3$; Figure 1)** This tree represents a "balanced" structure that ends up providing different advantages for both BUL and ACL in different parts of the tree.

Let us next discuss each of the goal-plan structures and how the performance of BUL-based and ACL-based agents compare.

Under a structure like $\mathcal{T}_1$, the agent basically has several options of comparable complexity to resolve the top-level goal $G$ in a certain world state. In $\mathcal{T}_1$ there are 20 options. However, most such options (17 in our example, plans $P_i'$) inevitably lead to failure. The benefit of using the ACL approach comes from the fact that the agent will decrease the probability of selecting each of those 17 failing plans as soon as they fail for the first time. In contrast, BUL requires multiple failed experiences of each of those "bad" top-level options before decreasing their probability of selection because each subgoal of each plan $P_i'$ must be stable before that $P_i'$ is updated. The ACL agent did indeed perform better in our experiments, in that it achieved better success rate earlier as shown in Figure 4(a). Observe that, eventually, both approaches will yield optimal performance.[6]

Let us now analyse the goal-plan tree $\mathcal{T}_2$ shown in Figure 3. Under such a structure, all successful executions are encoded in a complex plan, in our case plan $P$. Other options that the agent may have (e.g., plans $P_i'$) are of less complexity, but do not lead to solutions for resolving the goal.[7] Because the plan containing the solution, namely $P$, is fairly complex, there are many ways the agent may fail when exploring the decomposition of $P$. The agent needs to make several correct choices to obtain a successful execution. Although we expected BUL to yield better agent performance than ACL, the difference was enormous in our experiments. Figure 4(b) shows that while the BUL approach achieves optimal performance, which amounts to slightly over $40\%$ rate of success, in slightly more than $500$ iterations, the ACL scheme, requires more than $3000$ execution experiences. The reason is this: since there are more chances to fail plan $P$ initially, ACL marks this plan as "bad," along with the non-working plans $P_i'$. On the other hand, BUL does not treat any plans as "bad" until they are stable, so plan $P$ is explored with *equal* likelihood to plans $P_i'$. Eventually, the simpler $P_i'$ plans become stable first, leaving BUL to explore plan $P$ almost exclusively. This structure shows exactly the problem discussed in the previous section, namely, the drawbacks of assuming that a plan is a bad option just because it happened to fail, without consideration of the confidence in the choices made below it.

Finally, consider the hierarchical structure $\mathcal{T}_3$ depicted in Figure 1. In this case, the top-level goal $G$ has four relevant plans, which are all "complex," that is, they all have several levels and multiple goals and plans. However, given a world state, only one particular path in this hierarchy will lead to a successful execution (of course, for different states, different top-level plans may apply). Among other things, this means that at the top-level the agent needs to select the right plan given the current world state. All other plans are bound to eventually fail. We argue that this is a common feature found in many BDI agent applications, in that even though the agent has been programmed with several strategies for resolving a goal, each one is crafted to cover uniquely a particular subset of states. In other words, these are systems with low *know-how* overlap. With respect to the two learning approaches we are considering, structure $\mathcal{T}_3$ provides advantages for both of them, in different parts of the tree. The ACL scheme is expected to learn faster the inadequacy of the non-working top-level programs, whereas the BUL would better explore, and find a solution, within the working top-level plan. This balance is evident in Figure 4(c) where both approaches show comparable performance.

## Plan Applicability and Optimality

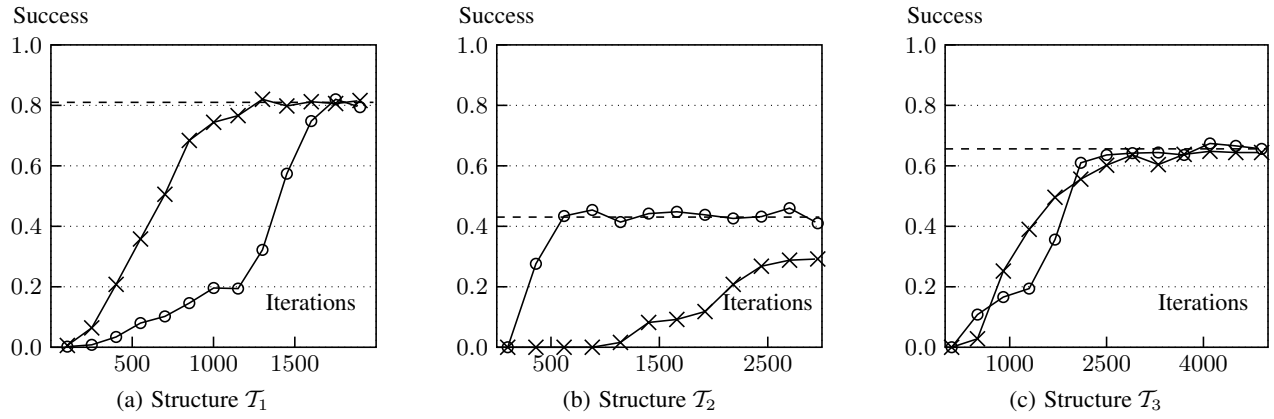
So far, we have assumed that the agent considers all relevant plans for a goal to be *applicable*, even though some may have a very low chance of success. This implies that, in contrast with standard BDI systems, our extended learning BDI framework will *always* select a plan from the relevant options. Because executing a plan is often not cost-free in real systems, it is likely that an adequate plan selection mechanism would in fact *not* execute plans with too low a probability of success. This in turn implies that the system may decide to fail a goal without even trying it, if it considers that the high likelihood of failure does not justify the cost of attempting any of the relevant plans. This is exactly what standard BDI systems do. When no applicable plan is found for a certain event-goal, that event-goal is failed right away.

To understand the impact of applicability in our BDI learning framework, we modified the probabilistic plan selection so that the agent does *not* consider plans whose chances of success are deemed below a threshold; in our case we set this threshold to $20\%$. For

as represented by the number of levels of abstraction and the numbers of goals at each level. The key factor is the number of abstraction levels—abstract plans are not in themselves complex.

[6]Optimal performance in this case amounts to a success rate of $81\%$, as the environment fails with probability .1 for every (working) action and each successful execution involves the performance of two actions (leaf plans consist of single actions).

[7]This is an extreme case for illustrative purposes. Of course the simpler plans $P_i'$ would, in a real program, lead to a solution in some world states or it would not make sense to include them. The same effect would however arise if most world states had solutions only in a complex plan branch.




(a) Structure $\mathcal{T}_1$ (b) Structure $\mathcal{T}_2$ (c) Structure $\mathcal{T}_3$

**Figure 4: Agent performance under BUL (circles) and ACL (crosses) schemes. Each point represents results from $5$ experiment runs using an averaging window of $100$ samples. The dashed line represents optimal performance (Note that outcomes are always $0$ or $1$ so more than expected consecutive successes may seems like "above" optimal performance when averaged).**

simplicity in this experiment, we removed the non-determinism in the environment: actions either fail or succeed in each world state.

Using the structure $\mathcal{T}_3$ we found that whereas the BUL scheme maintains its performance (and in fact may slightly improve due to truly failing leaf plans being ruled out earlier), the ACL approach may not learn at all and end up eventually failing the top-level goal *always*. This is reported in Figure 6 (dotted lines).

The explanation for this undesirable behavior under ACL is as follows. Initially, the agent tries all top-level plans for the top-level goal, including the ones containing potential successful executions. Because of their complexity, the chance of finding a successful execution immediately is very low, and most executions fail initially. With each failure, ACL decreases the feasibility of all plans tried, including the top-level one. After several failures, all plans for the top-level goal eventually go below the applicability threshold of the system (including the "good" plans). When that happens, the system has no more applicable plans for the goal and will therefore fail it always. This behavior does not arise in the original system, because even if all plans perform very poorly, the agent would always pick one anyway, the successful path would eventually be found, and the context decision trees of the plans associated with such a path would then start "recovering."

The reason BUL exhibits more robust behaviour here is that false negative executions (i.e., failing executions of plans that do encode successful runs) will *not* be recorded. The BUL approach relies on a *confidence* test—stability—that checks whether we are sufficiently well informed to trust that the current failure is indicative of future attempts. In the next section, we explore an alternative approach to confidence that takes account of how sure we are of the decision tree when we use it, rather than using stability as a confidence measure for deciding when to record. Whereas stability is a boolean measure (true or false), the alternative measure gives us a more fine-grained *degree* of confidence.

# 6. INFORMED PLAN SELECTION

Our new approach relies on the idea that confidence in a plan's decision tree increases as more of the possible choices below the plan in the goal-plan structure are explored.

So, with each plan in the goal-plan tree hierarchy, we identify its set of potential *choices* as the set of all potential execution paths *below* the plan in the hierarchy. This can easily be computed offline. Intuitively, a plan's decision tree is more *informed* for a world state if it is based on a larger number of choices having been explored in that state. We say that a plan has a higher degree of *coverage* as more of its underlying choices are explored and accounted for in the corresponding decision tree. Technically, given a decision tree $T$ for a certain plan, we define its coverage for the world state $w$ as $c_T(w) \in [0, \ldots, 1]$. Initially, when the plan has not yet been executed in a world $w$, its coverage in such state is $c_T(w) = 0$ and the agent has no basis for confidence in the likelihood of success estimated by $T$ for $w$. As the different ways of executing the plan in the world state $w$ are explored, the value of $c_T(w)$ approaches $1$. When all choices have been tried, $c_T(w) = 1$ and the agent may rely fully on the decision tree estimation of success. In this way, coverage provides a confidence measure for the decision tree classification.

We then construct a probabilistic plan selection function that includes the coverage-based confidence measure. Formally, we define the plan selection weight $\Omega'(w)$ as a function of the decision tree determined success expectation $p_T(w)$ and the degree of coverage $c_T(w)$:

$$\Omega'_T(w) = 0.5 + [c_T(w) * (p_T(w) - 0.5)] .$$

Initially the selection weight of the plan for a previously unseen world state $w$ takes the default value of $0.5$. Over time, as the various execution paths below the plan are tried in $w$, its coverage increases and the selection weight approaches the true value estimated by the plan's decision tree.

Each time a plan execution result is recorded, the coverage $c_T(w)$ for a world $w$ is calculated and stored. It requires, in principle, $\tau \times |S|$ *unique* executions of a plan for it to reach *full* coverage, where $\tau$ is the total number of choices below the plan and $|S|$ is the number of possible worlds. Practically, however, it takes significantly less since choices below a plan are effectively an AND/OR tree, and each time an AND node fails, the subsequent nodes are not tried and are counted as covered for the world in question. Also, a plan is generally not executed in every world state, so in practice it will only need to be assessed in the subset of the world states that is relevant to it.

We are now ready to revisit the two learning approaches ACL and BUL from the previous section, but this time using the modified selection weighting based on coverage. We will refer to the new ap-

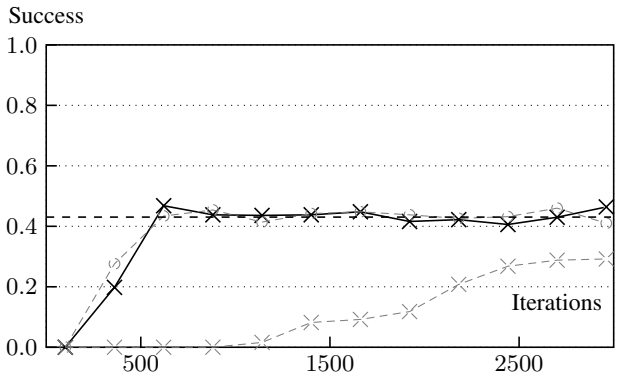



**Figure 5: Performance of ACL+$\Omega'$ (solid crosses) in structure $\mathcal{T}_2$ compared against the earlier results for ACL+$\Omega$ and BUL+$\Omega$ (both in dotted grey).**

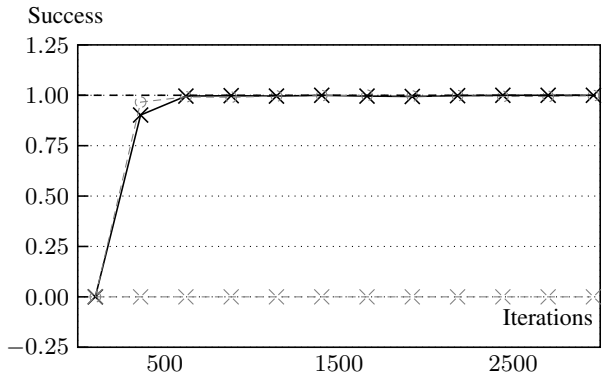



**Figure 6: Performance of ACL+$\Omega'$ (solid crosses) compared against ACL+$\Omega$ and BUL+$\Omega$ (both in dotted grey) in structure $\mathcal{T}_2$ using an applicability threshold of $0.2$.**

proaches as ACL+$\Omega'$ and BUL+$\Omega'$, respectively. Similarly, ACL+$\Omega$ and BUL+$\Omega$ correspond to the approaches using the *original* selection weighting that only uses its decision tree success expectation, that is, $\Omega_T(w) = p_T(w)$.

Our first observation is that the BUL+$\Omega$ and BUL+$\Omega'$ approaches show similar performance. This is not surprising, as the stability test performed by these agents at each plan node inherently results in close to full coverage. Indeed, for a plan to become "stable," the agent needs to (substantially) explore all possible ways of executing it. The stability check, then, effectively reduces $\Omega'_T(w)$ to $\Omega_T(w)$. So, for simplicity, we shall not give a further account of the BUL+$\Omega'$ approach in this section.

We now focus on the ACL approach. For the ACL-favouring structure $\mathcal{T}_1$, we find that the performance of ACL+$\Omega'$ matches that of ACL+$\Omega$ reported earlier in Figure 4(a). Similarly, for the balanced structure $\mathcal{T}_3$ where previously both ACL and BUL performed equally well, the performance of ACL+$\Omega'$ was the same as that reported for ACL+$\Omega$ earlier in Figure 4(c). Thus, for the cases where ACL+$\Omega$ was performing reasonably well, the new ACL+$\Omega'$ approach maintains comparable performance.

The benefit of the coverage-based approach is apparent, though, when one considers the goal-plan structure $\mathcal{T}_2$ in which the ACL+$\Omega$ performed poorly (cf. Figure 4(b)). Here, the ACL+$\Omega'$ scheme showed a dramatic improvement over ACL+$\Omega$. Figure 5 shows this change with the results for the new approach to plan selection ACL+$\Omega'$ superimposed over the original results from Figure 4(b). The reason why the new plan selection mechanism improves the ACL learning scheme is that even though the success estimation $p_T(w)$ for a given plan $P_i$ would still be low initially (remember that ACL, in contrast with BUL, would record all initial failure outcomes for $P_i$), the agent would not be very confident in such estimation until the plan's coverage increases; therefore the selection weight $\Omega'_T(w)$ will initially bias towards the default weight of $0.5$. In other words, the false negative outcomes collected by the agent for plan $P_i$ would not be considered so seriously due to low plan coverage. As full coverage is approached, one would expect the agent to have discovered the success execution encoded in $P_i$.

Even more interesting is the impact of the new plan selection mechanism on agents that work with an applicability threshold, i.e., agents that may not select plans that are deemed unlikely to succeed. Here, the original ACL+$\Omega$ approach completely fails, as it collects many negative experiences early on, quickly causing plans' success expectation to fall below the selection threshold. For ACL+$\Omega'$, even if a plan is deemed with very low expectation of success, its selection weight would be biased towards the default value of $0.5$ if it has not been substantially "covered." Hence, provided that the applicability threshold is lower than the default plan selection weight, then ACL+$\Omega'$ is indeed able to find the solution(s). Figure 6 shows the ACL+$\Omega'$ performance in goal-plan structure $\mathcal{T}_2$ for an applicability threshold of $20\%$.

The above results show that the coverage-based confidence weighting can improve the performance of the ACL approach in those cases where it performed poorly due to false negative experiences, i.e., failure runs for a plan that includes successful executions. Furthermore, coverage provides a flexible mechanism for tuning agent behaviour depending on application characteristics. Consider equation $\Omega'_T(w)$ with the coverage term modified to $c_T(w)^{1/\alpha}$, with parameter $\alpha \in [0, \ldots, \infty)$. Interestingly, as $\alpha \approx 0$, ACL+$\Omega'$ will behave more like BUL+$\Omega$: $c_T(w)^{1/\alpha}$ transitions directly from 0 to 1 when $c_T(w)$ reaches 1 (and remains zero otherwise). On other hand, when $\alpha \approx \infty$, ACL+$\Omega'$ will behave more like the ACL+$\Omega$: $c_T(w)^{1/\alpha}$ transitions from 0 to 1 faster and $\Omega'(w) \approx p_T(w)$. With $\alpha = 1$ we get our initial equation. It follows then that ACL+$\Omega'$ provides a *middle ground* between the ACL+$\Omega$ and BUL+$\Omega$ schemes.

Finally, we note that coverage-based selection weights encourage the agent to explore all available options. This further ensures that all solutions are systematically found, allowing the agent to decide which solution is optimal faster. For some domains this may be an important feature.

# 7. DISCUSSION AND CONCLUSION

In this paper, we proposed a technique to enhance the typical plan selection mechanism in BDI systems by allowing agents to learn and adapt the context conditions of plans in the agent's plan library. As designing adequate context conditions that take full account of the agent's environment for its complete life-cycle is a non-trivial task, a framework that allows for the *refinement* of (initial) context conditions of plans *based on online experience* is highly desirable. To this end, we extended the typical BDI programming framework to use decision trees as (part of) plan's context conditions and provided a probabilistic plan selection mechanism that caters for both exploration and exploitation of plans. After empirically evaluating different learning strategies suitable for BDI agents against various kinds of plan libraries, we concluded that an aggressive learning approach combined with a plan selection scheme that uses a confidence measure based on the notion

of plan coverage is the best candidate for the general setting. The work carried out here is significant for the BDI agent-oriented programming community, in that it provides a solid foundation for going beyond the standard static kind of BDI agents.

The framework presented here made a number of simplifying assumptions. We did not consider the effects of conflicting interactions between subgoals of a plan. In fact, the way a subgoal is resolved may affect how the next subgoal can be addressed or even if it can be resolved at all. Our current approach will not detect and learn such interactions; each subgoal is treated "locally." To handle such interactions, the selection of a plan for resolving a subgoal should also be predicated on the goals higher than the subgoal, that is, it should take into account the "reasons" for the subgoal. Similarly, we did not consider the effects of using goal failure recovery, under which alternative plans for a goal are tried upon the failure of a plan. Also, we have only dealt with domains described via boolean propositions. To handle continuous attributes (e.g., discretize *temperature*), our approach requires that either these attributes are discretized (e.g., *cold*, *warm*, and *hot*) or additional discrete attributes be used to test the continuous ones (e.g. *temperature* $< 25.2$).

One critique of the coverage-based confidence measure used is that it has a defined end state, namely $c_T(w) = 1$. In a real system, however, learning and re-learning will occur indefinitely, as the agent continually tries to *adapt* to a changing environment. This implies that an agent's confidence in a decision tree's classification would also require calibration when the environment has changed. If the change was deliberate, then our confidence could be reset and subsequently *re-built*. Without such an explicit signal, the agent must rely on other methods for determining when the environment has changed significantly. An appealing measure for recognising environmental changes is through the relatedness of its features. For instance, an observation that the grass is *wet* may have a high correlation to the fact that it is *raining*. If at some point, the agent were to witness a world where it is not raining but the grass is indeed wet (for some other new reasons), then this world would be "atypical," and as a result, the agent may have reason to reduce its confidence in a plan's decision tree classification of this new world. It turns out that efficient algorithms exist—some already included in the weka library—that perform inference in and learning of Bayesian networks [10], which the agent can appeal to in building a model of the environment for the purposes just described.

The issue of combining learning and deliberative approaches for decision making in autonomous systems has not been widely addressed. In [13], learning is used *prior to deployment* for acquiring low level robot soccer skills that are then treated as fixed methods in the deliberative decision making process once deployed. Hernández et al. [8] give a preliminary account of how decision trees may be induced on plan failures in order to find an alternative logical context conditions in a deterministic paint-world example. More recently, [18] proposes a method for learning hierarchical task network (HTN) method preconditions under partial observations. There, a set of constraints are constructed from observed decomposition trees that are then solved *offline* using a constraint solver. Despite HTN systems being automated planning frameworks, rather than execution frameworks, these are highly related to BDI agent systems when it comes to the *know-how* information used—learning methods' preconditions amounts to learning plan's context conditions. In constrast, in our work, learning and deliberation are fully integrated in a way that one impacts the other and the classical exploration/exploitation dilemma applies.

This paper extends our earlier work [1] in several ways. First, our conservative learning approach based on the notion of plan "stability," is substantially more grounded than in [1], where a plan is just required to be executed a *fixed* number of times for failure executions to be recorded. Second, only one goal-plan hierarchical structure was used for experimentation in [1]; here we considered different structures identifying various types of plan libraries. More importantly, we explored the realistic case of agents with plan applicability thresholds. Contrary to what was implied in [1], our extended work here suggests that some sort of *confidence* test is indeed worthwhile—this was the motivation behind our coverage-based approach.

## Acknowledgments

We thank the anonymous reviewers for their helpful comments. We also acknowledge the support of Agent Oriented Software and the Australian Research Council (under grant LP0882234).

# 8. REFERENCES

[1] S. Airiau, L. Padgham, S. Sardina, and S. Sen. Enhancing adaptation in BDI agents using learning techniques. *International Journal of Agent Technologies and Systems (IJATS)*, 1(2):1–18, Jan. 2009.

[2] S. S. Benfield, J. Hendrickson, and D. Galanti. Making a strong business case for multiagent technology. In *Proceedings of AAMAS*, pages 10–15. ACM Press, 2006.

[3] R. Bordini, J. Hübner, and M. Wooldridge. *Programming Multi-agent Systems in AgentSpeak Using Jason*. Wiley Series in Agent Technology. Wiley, 2007.

[4] M. Bratman, D. Israel, and M. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355, 1988.

[5] P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas. JACK Intelligent Agents: Components for intelligent agents in Java. *AgentLink News*, 2:2–5, 1999.

[6] M. Dastani. 2APL: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, June 2008.

[7] M. P. Georgeff and F. F. Ingrand. Decision making in an embedded reasoning system. In *Proceedings of IJCAI*, pages 972–978, Detroit, USA, 1989.

[8] A. Guerra-Hernández, A. E. Fallah-Seghrouchni, and H. Soldano. *Learning in BDI Multi-agent Systems*, volume 3259 of *LNCS*, pages 218–233. Springer, 2004.

[9] K. Hindriks, F. D. Boer, W. V. D. Hoek, and J. Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

[10] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[11] D. Morley and K. L. Myers. The SPARK agent framework. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 712–719, 2004.

[12] M. E. Pollack. The uses of plans. *Artificial Intelligence Journal*, 57(1):43–68, 1992.

[13] M. Riedmiller, A. Merke, D. Meier, A. Hoffman, A. Sinner, O. Thate, and R. Ehrmann. Karlsruhe brainstormers - a reinforcement learning approach to robotic soccer. In *RoboCup 2000: Robot Soccer World Cup IV*, 2001.

[14] E. Swere, D. Mulvaney, and I. Sillitoe. A fast memory-efficient incremental decision tree algorithm and its application to mobile robot navigation. In *Proceedings of IROS*, 2006.

[15] J. Thangarajah, M. Winikoff, L. Padgham, and K. Fischer. Avoiding resource conflicts in intelligent agents. In *Proceedings of ECAI*, pages 18–22, 2002.

[16] P. E. Utgoff, N. C. Berkman, and J. A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, 1997.

[17] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[18] H. Zhuo, D. Hu, C. Hogg, Q. Yang, and H. Munoz-Avila. Learning HTN Method Preconditions and Action Models from partial Observations. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, 2009.