Elsevier Editorial System(tm) for Robotics and Autonomous Systems
Manuscript Draft

Manuscript Number: ROBOT-D-09-00261R1

Title: Extending BDI Plan Selection to Incorporate Learning from Experience

Article Type: SI:Hybrid Control for Autonomous Systems

Keywords: BDI Intelligent Agents, Machine Learning, Hybrid Systems

Corresponding Author: Mr Dhirenda Singh,

Corresponding Author's Institution: RMIT University, Melbourne, Australia

First Author: Dhirenda Singh

Order of Authors: Dhirenda Singh; Sebastian Sardina; Lin Padgham

Abstract: An important drawback to the popular Belief, Desire, and Intentions (BDI) paradigm is that such systems include no element of learning from experience. We describe a novel BDI execution framework that models context conditions as decision trees, rather than boolean formulae, allowing agents to learn the probability of success for plans based on experience. By using a probabilistic plan selection function, the agents can balance exploration and exploitation of their plans. We extend earlier work to include both parameterised goals and recursion and modify our previous approach to decision tree confidence to include large and even non-finite domains that arise from such consideration. Our evaluation on a pre-existing program that relies heavily on recursion and parameterised goals confirms previous results that naive learning fails in some circumstances, and demonstrates that the improved approach learns relatively well.

Extending BDI Plan Selection to Incorporate Learning from Experience (Responses to Reviewers)

Dhirendra Singh & Sebastian Sardina & Lin Padgham

RMIT University, Melbourne, Australia

Abstract

This document details the responses and changes based on reviewers comments on the original manuscript submitted to the Special Issue on Hybrid Control of Autonomous Systems of the Journal of Robotics and Autonomous Systems [3].

General Comments

We would like to point out that our previous work referred to as "under review" in the manuscript is now accepted as a peer-reviewed publication [2] and we have duly updated the citation. Some concerns raised by the reviewers have also been addressed in [2] and we have noted this in our responses where appropriate.

Please note that the page number and Figure references used in this document refer to the revised manuscript and may be different from those in reviewer comments.

Email address: {dhirendra.singh,sebastian.sardina,lin.padgham}@rmit.edu.au (Dhirendra Singh & Sebastian Sardina & Lin Padgham)

1. Reviewer #1 Comments and Responses

1.1. The problem is well motivated and is clearly relevant to the topic of this special issue. The way the above two measures are computed seems plausible, although this is only partly supported by the experiments. This is because of the "optimizations" you mention on page 16 for the particular problem at hand. What would happen if you did not use these optimizations?

We see how the details of the optimisations have added to some confusion. These optimisations were in place for *all* experiments and so impacted both approaches — the baseline ACL and the improved $ACL + \Omega$. Our main motivation for this was to speed-up absolute convergence times for both approaches during experimentation allowing us to significantly reduce the run-time for our full suite of experiments. The impact of not applying the optimisations is that the graphs of Figure 5 and Figure 6 take more iterations to converge to the solutions. This however does not impact any of our claims that only focus on the effectiveness of the new $ACL+\Omega$ approach "relative" to the baseline ACL.

For that reason we have removed the paragraph on optimisations (was paragraph before start of [4.2 Results] on page 15), as it is not relevant to the discussion and does not bear on the relative results. **Action**

1.2. You say on p. 12 that δ_{Pt} can be computed offline. It is not clear to me how. In your example, you state the equation for $\delta_P t$ but not how you obtained it. Was it derived automatically or by hand?

For this work, we have calculated δ_{Pt} in terms of average breadth and depth of the structure, where the depth of a structure with recursion is the maximum level of recursion, to provide an approximation of the complexity of the structure. Of course, there are other ways of calculating δ_{Pt} (e.g., in [2] we have used an accurate calculation of the number of choices below plans; however this is not feasible anymore when goals and plans are schemas with possibly infinite instances); the whole idea is to somehow measure the complexity of a hierarchical structure.

We have added a footnote to the first paragraph on page 12 to clarify that δ_{Pt} is computed in terms of breadth and depth of a structure but that that other measures may be used with analogous overall results. **Action**

1.3. *You only compare your new approach to your previous work in the experiments. Why did you not include a comparison with the existing BDI program that comes with JACK? This is all the more surprising since you yourself ask: does our learning framework achieve the performance of the existing system. Does the existing program always do the right thing? In any case, it would be good to compare at least some of the hand-crafted context formulas with those learned by the DT approach.*

The original JACK program has perfectly crafted context conditions that work precisely so that all *Solve* events are always successfully resolved (equivalent to 100% in Figure 5a). While the structured exploration of $ACL + \Omega$ succeeds in resolving the *Solve* goal for all possible configurations (new Figure 4b crosses) by $12k$ iterations, it nonetheless does not achieve perfect performance (converges to about 90% performance in Figure 5a). The reason is that the decision tree is a compact representation that does not guaranteed 100% correctness when classifying the training data (we discuss this inherent trade-off in [3.1 Integrating Decision Trees into Context Conditions for Plans]). For instance, a plan that has previously succeeded in a given world may not get selected because the decision tree returns a low probability of success due to misclassification of the related training sample. The only way to guarantee correct selection would be to use the training data directly, for instance using a look-up table.

We would further like to point out here that the learnt context conditions do not "look" like the original ones. When the learnt decision trees are converted to rules, they are quite complex and perhaps not intuitive as the original conditions are. This is mainly due to representational differences. For instance, the real context conditions are relational while the learnt ones are propositional.

We have updated the final paragraph of [4.2 Results] on page 18 to clarify this point.

Action

1.4. *The paper is well written and the relevant literature is discussed/cited as far as I can tell. (I am no BDI expert.)*

Please note that we have significantly improved the related work discussion in [5. Discussion and Conclusion] on page 20.

Action

1.5. Typo: p.20: preiously

This has now been corrected.	**Action**

2. Reviewer #2 Comments and Responses

2.1. The overall work seems reasonable and appears to be effective, although it does not seem to be particularly challenging. The learning approach is limited by the inabilitiy to take into account goal inter-dependencies. Nevertheless, the work represents progress towards endowing BDI agents with learning capabilities.

Although the problem may initially appear non-challenging, it indeed is. There are many nuances that come up mainly due to the fact that *(i)* the objects to be learnt (i.e., plan's context conditions) are strongly inter-related, what happens in one plan is affected by what happens in the plans below; *(ii)* learning and acting happens interleaved; and *(iii)* there are typically many more failures than successes.

Providing a *principled* exploration mechanism of the several alternatives an agent may have in its know-how structure turns out to be non trivial, which explains why almost no work has been done on the topic.

With respect to the issue of inter-dependencies, we point out that this is not a limitation of our learning approach itself, but of the overall BDI programming paradigm. Addressing this would require a substantial modification in the BDI programming style in terms of representation, which is out of the scope of this paper. Here we aim to provide a learning account compatible with standard existing BDI architectures.

We have updated paragraph 4 on page 20 to clarify this point.	**Action**

2.2. In sec.3, 1st para., what do you mean by "event-goal type"? just a parametrized event-goal?

Yes. We have updated the paper to be consistent and use *parameterised* instead.	**Action**

2.3. The whole section 3.1 is written in a confusing way.

We have now significantly reworked this section to help improve understanding.

Action

2.4. In the 2nd para. pg 9, you talk about "the plan". One can guess that you mean learning is applyied for each plan in the library. But it would be better to make it explicit here, e.g., that you need a training data set for each plan in the library.

We have incorporated your suggestion and updated the last paragraph on page 8. We have also made this clear in the first paragraph of [2.2 Learning for BDI Plan Selection] on page 6.

Action

2.5. At the end of this paragraph, it says that a classification is learnt "based purely on the subset of attributes in w that are relevant to the contex condition of the plan." This is not clear. What 'w' are you referring to here? The next para. starts again talking about "The attributes in w...", what 'w'?

We have prefixed "world state" to w throughout and ensured that w is explained clearly at the start of the paragraph when we first mention it (last paragraph of page 8).

Action

2.6. Also in the 2nd para. it says "the decision tree will learn a classification..." isn't the tree itself what is being learnt?

We have reworded the last paragraph of page 8 so it is not confusing to the reader.

Action

2.7. In the 3rd para. pg 11, it says that "...we may never find the bottom or leaf nodes. This has implication for any bottom-up strategies." You mean top-down strategies?

We have rewritten paragraph 1 on page 11 to remove any confusion.

Action

2.8. *In sec. 3.2, goals G are sometimes followed by square brackets, as in $G[\phi]$, and sometimes buy angle brackets, $G\langle\phi\rangle$, which one may guess represent modalities, but the notation has not been defined. So the reader is left guessing what the notation means.*

We apologise for the confusing notation. These are not modalities. $G[P : w]$ states that plan P is tried on world w to resolve goal G. We have changed $G\langle\phi\rangle$ to $G(\vec{x})$ which denotes goal G with parameters $\vec{x}$ (e.g., $clean(room1)$).

We have clarified and fixed the notation in Section 3.2. We now just use $G[P : w]$ and $G(\vec{x})$.

Action

2.9. *Sec 3.4, 1st para., you refer to a "previous definition of the learning task (Section 3)", which one can guess refers to the informal definition at the beginning of Section 3, since there is no definition anywhere. Btw, the reference itself is in section 3.*

We have removed the reference and reworded the first paragraph of [3.4 Handling Parameterised Event-Goals] on page 12.

Action

2.10. *Also in sec 3.4, you define the training data set as tuples [w U phi, o]. This is strange since I would think the world would be represented differently from a goal's parameters, but you specify the first element in a training data tuple as the union of a world and the goal's parameters. Why not add a ground goal instead? Or use a triple?*

Thank you for picking this. We have now updated the last paragraph of [3.4 Handling Parameterised Event-Goals] on page 13 to now use a triple.

Action

2.11. *At the end of sec. 4.1, you discuss some "optimizations" that are possible in the case of the towers of hanoi problem. It is not really clear why you need to make these optimizations (I'd called them simplifying assumptions rather than optimizations). What if you don't make them?*

We refer the reviewer to the response in 1.1 that addresses this concern.

Typos etc:

2.12. pg 8: "a plans ..." → "a plan ..." or "a plan's ..." (occurs in many places).

We have reviewed the manuscript and corrected these cases. **Action**

2.13. pg 20: "preiously learnt"

This has now been corrected. **Action**

3. Reviewer #3 Comments and Responses

3.1. The major weaknesses of the paper is the lack of emperical or formal results and the number of assumptions made which makes the general applicability questionable.

In general, we believe that while the assumptions mean that the current framework is not ready for practical systems, it nonetheless is a significant step forward in our ongoing work to that end. Please refer to our responses in 3.14 and 3.16 that address specific questions that are related to this comment.

3.2. The paper is well written and is easy to understand but some parts in the sections 2 and 3 would be easier to understand if a good real example would guide the line of argumentation (e.g. an example from the agents or robotics domain). ... A good example of a domain used in the explanation of the background in section 2 and in the development of the approach in section 3 would ease the understanding of the paper and would even non-bdi experts to profit from the paper.

We take the point and agree it may be hard for non-bdi experts to get a good understanding of the BDI approach by reading Section 2. In particular, it was difficult to see what plan libraries built from plan-rules, in turn built from events and actions, were.

We have added a short example of plan rules that would arise in an elevator controller. Please see second paragraphs 2 and 3 of [2.1 BDI Agent Systems] on page 4. **Action**

3.3. In the middle part of page 7 the authors claim that event types and recursion is important to handle. But it is not very well explained why these constructs are important to handle.

Event types are necessary to be able to compactly represent the domain know-how information by providing "schemas" of plans that can be instantiated with multiple objects. Recursion is very important as it allows, in these languages, to express (unbounded) iteration (as in logic programming, there is in general no imperative while or loop constructs, thus recursion becomes the main construct for iteration).

Hopefully the plan rules example for an elevator controller introduced in [2.1 BDI Agent Systems] will help clarify why schematic plan rules and recursive calls in programs are important. We have also updated paragraph 2 on page 7 ("A limitation...") to be more clear on the importance of handling these 2 features.

Action

3.4. Also throughout the paper it is not really point out how the event types are handled in fact.

We handle event types in the learning framework by adding the event parameters as attributes to the decision trees' training data. Please see the last paragraph of [3.4 Handling Parameterised Event-Goals] on page 13 for more information on this.

3.5. The term "coverage" is used very often starting with page 7. It seems to be an important part in the approach but is neither formally defined nor really explained. A definition and a deeper explanation would help the reader to understand the approach.

We consider the reliability of a plan's decision tree in a given world state to be proportional to the number of sub-plan choices (or paths below the plan in the goal-plan hierarchy) that have been explored in that world state. *Coverage* refers to the set of explored paths relative to the set of all possible paths. The greater the coverage, the more we have explored and the greater the confidence in the resulting decision tree.

We have now updated paragraph 1 on page 7 with this definition.

Action

3.6. *On page 9 the authors introduce decision trees to represent the chance of a plan to be successful which is used later to define if a plan should be used in a certain situation. To me it is not clear as a decision tree covers both the negative and the positive case of the execution how the information on the negative outcome is used.*

Yes, the decision tree classification is concrete i.e. either *success* or *failure*. However, since the decision tree induction usually has some trade-off between accuracy of classification and compactness of representation, then some training samples get incorrectly classified in the wrong "bucket" (where the bucket name is *success* or *failure*) when infact the actual outcome class of those training samples is different. So in a given bucket (or class) one may get a total of m training samples out of which n are misclassified. The ratio $1-(n/m)$ gives the likelihood of class membership and is automatically provided by the *weka* package. This is the fraction we refer to when we talk about the "chance" of success (or failure).

We agree that this was not well explained. We have updated the last paragraph in [3.1. Integrating Decision Trees into Context Conditions for Plans] on page 9 to make this clear. **Action**

3.7. *On the second half of page 9 the authors claim that the set of attributes used in the learning of a decision tree are selected by the programmer of the domain. This means in the first place additional work for the programmer and might in the second place prevent the system to find a good solution. Would it be feasible to learn the conditions without this artificial limitation of the attributes?*

We believe the requirement for the user/programmer to state relevant domain propositions is a realistic one. It is generally feasible for the domain expert to state which features of the world might play some role when solving a given goal. For example, in the travelling domain, airlines schedules and location of airports are relevant for the goal of buying an airline ticket, whereas domain propositions describing the humidity or features of car rental companies are mostly not relevant. See that the requirement is for the domain expert to enumerate the relevant features for each goal; this is generally a feasible request and much less demanding than requesting exact plans' preconditions.

Nonetheless, relevant features could be automatically extracted from the plan library if domain information is available for primitive actions. For example, if in addressing some goal, it might happen that action “drive(src,dest)” is executed, then the fuel level, being mentioned in the precondition of the action, should be considered as “relevant” for the goal. Though this automatic extraction is not the subject of this paper, we think that in a full-fledged learning BDI system, both automatic analysis and domain expert knowledge would be used to estimate the relevant set of propositions for a goal.

We have added a footnote to the paragraph 2 of page 9 to highlight this point.

Action

3.8. On page 10 the authors explain recursion and the need for. Could you give a proper example for a recursive event goal in a real world domain.

As in logic programming, recursive calls (in our context, recursive “subgoaling”) are necessary to express iterative procedures. Please see our response in 3.3 above.

3.9. In chapter 3.3 the authors explain that the training set is build up incrementally. To my understanding it might be the case that the training set so far and the plan selection learned so far has an influence of the future training data. Therefore, is it possible that a somehow unlucky plan selection and therefore probably bad training set prevent the system to converge to a good or correct set of conditions?

Yes, absolutely correct. This concern when performing *online* learning (learning while you act) has been central in our previous work as it is here. In the first instance [1] we look at how the notion of informed decisions may be used to do selective recording of results in order to filter out misleading training data. In the second [2] we explore a confidence measure that may be used to decide how much value we will put in the resulting decision tree. These are two orthogonal approaches i.e. how best to collect training data and how best to use it.

We have updated paragraph 2 of [2.2. Learning for BDI Plan Selection] on page 6 to include this point about ongoing learning impacting future learning.

Action

3.10. Formula 2 on page 14. The construction of the formula looks to me a little bit ad-hoc. Is there a better justification for the formula? What are others or maybe better weight functions.

The formula is the same as that described in [2] where we also discuss an alternative weighting and how it may be used to adjust the performance of the system to suit different hierarchies. Considereing that this issue has been previously discussed, and due to lack of space here, we have chosen not to include this justification in this paper.

However, we have added a footnote to paragraph 2 of [3.5. Calculating Plan Selection Weights based on Condence] on page 13 referring the reader to the origin of the formula.

Action

3.11. In the middle part of page 16 the authors state that they do not continuous exploration once a solution is found. Why this is done or why it is sufficient? It seems to me that this assumption is against the authors arguments for the properties of their approach.

We are sorry this has been a cause of much confusion. We refer the reviewer to the response in 1.1 that addresses this concern.

3.12. In 4.2. k is introduced but no real number for it in the experiments is presented nor the influence of this parameter is described. Please give more information on this.

The parameter k controls the rate of change of decay δ_{Pt} and is currently arbitrarily set to 4.0. Since the decay δ_{Pt} determines our confidence in the decision tree, then k determines how quickly or slowly we will get there in terms of the number of steps.

We have updated paragraph 1 of [4.2 Results] on page 16 with this information.

Action

3.13. *At the end of page 18 the authors that the applicability threshold is set to 20% in the second experiment. It seems to my a little bit artificial and tuned towards an ad-hoc justification for the approach. Please give further information on the performance of the compared algorithms in the context of this threshold.*

We agree that the threshold setting of 20% is somewhat arbitrary, however it is consistent with that used in [2]. The difference between the default weight (0.5) and the cut-off weight (threshold of 0.2 in this case) determines how much "give" we have in the exploration. The closer the threshold is to the default, the higher the likelihood that the agent will give up before achieving the goal.

We have added a footnote to paragraph 1 on page 18 to help clarify this further. **Action**

3.14. *On the top of page a number of assumptions are stated. These assumption seems to limit the practical general applicability of the approach a lot. Please give a deeper justification for these assumptions and on probably solutions to them.*

We assume the reviewer is referring to the assumptions on page 20 of the original submission. We have improved justification of assumptions in [5. Discussion and Conclusion] on page 19. We have also emphasised that while more work is required before our framework is ready for use in practical systems, this work nonetheless constitutes significant progress in this direction. **Action**

3.15. *The coverage of related research seems to me a bit weak. Only three references on such a topic is not much. I guess there is more related work out there. Please rework these section.*

We have reworked the entire related work coverage in [5. Discussion and Conclusion] and now provide a total of nine related references on pages 20. **Action**

3.16. *The result section is not very much convincing. First to me it is not clear why a simple toy domain like the towers of Hanoi justifies the usefulness of the algorithm. Even the two experiments in this domain are made in a ad-hoc manner. For a journal this is too weak. I like to see a deeper treatment of the toy domain and to see experiments in a more advanced, more relevant or even real world domain. Otherwise I am not completely convinced about the advantages of the approach.*

We acknowledge your overall concern, but hope to strongly convey that the chosen domain (taken from the JACK distribution example) is in fact justified as it captures important characteristics that must first be treated and understood in a principled way in an idealised setting that is not possible by directly adopting a real world example. We reiterate that our end goal *is* to allow use of our learning framework in real applications. We further refer the reviewer to our response in 2.1.

Our experiments were chosen to convey the key benefits of the $ACL + \Omega$ approach using a similar experimental setup as previously [2]. In fact, our testing for this work includes all our earlier experiments (albeit lacking parameterised events and recursion) to ensure that the new approach does not invalidate our earlier work.

The first experiment (Figure 5) captures some key points that we wish to highlight. Firstly, it shows the benefits of the structured exploration of $ACL + \Omega$ over the baseline ACL approach [1, 2]. The result also confirms the merits of the exploration-based confidence idea that in it's original form [2] did not scale to recursive programs and formed the motivation for this work. Secondly, the experiment reinforces the intuition that the structured $ACL + \Omega$ approach is likely to show more gains where the solution is more complex. For instance, for solutions requiring five levels of recursion, ACL achieves only 50% success by the end of the experiment at $5k$ iterations whereas $ACL + \Omega$ achieves 95% success within $3.5k$ iterations. For levels three and one, the advantages of $ACL + \Omega$ over ACL are progressively less.

The first experiment equips us to understand the performance of the two approaches in the full experiment shown in Figure 6a. We have now added a second view of the results of experiment two in Figure 6b that shows the progressive discovery of solutions for each approach. Here the benefit of $ACL + \Omega$ is clear. Figure 6b shows that $ACL + \Omega$ resolves all 52 goals within $12k$ iterations whereas ACL resolves only 47 by the end of the experiment at $20k$. At a similar point of comparison, to resolve 47 goals $ACL + \Omega$ takes around 6.4k iterations whereas ACL takes more than twice as long at 17.7k. Perhaps it is the choice of the third experiment with applicability thresholds that seems ad-hoc to the reader in this context, however it is there to emphasise that the improved $ACL+\Omega$ avoids this issue in a manner (of experiment) similar to the coverage approach [2] it borrows from.

We have added Figure 5b and significantly reworded [4.2 Results] to strengthen our case.

Action

Final Comments

We thank all the reviewers for their invaluable comments and the opportunity to address the points raised. We hope to make a valuable contribution to the Journal of Robotics and Autonomous Systems.

[1] S. Airiau, L. Padgham, S. Sardina, and S. Sen. Enhancing Adaptation in BDI Agents Using Learning Techniques. In *International Journal of Agent Technologies and Systems* (IJATS), 1(2):1-18, January 2009.

[2] D. Singh, S. Sardina, L. Padgham, S. Airiau. Learning Context Conditions for BDI Plan Selection. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, Toronto, Canada, 2010. To appear.

[3] D. Singh, S. Sardina, L. Padgham. Extending BDI Plan Selection to Incorporate Learning from Experience. Submission under review for the Special Issue on Hybrid Control of Autonomous Systems (HYCAS), Journal of Robotics and Autonomous Systems, 2010.

Extending BDI Plan Selection to Incorporate Learning from Experience

Dhirendra Singh & Sebastian Sardina & Lin Padgham
RMIT University, Melbourne, Australia

Abstract

An important drawback to the popular Belief, Desire, and Intentions (BDI) paradigm is that such systems include no element of learning from experience. We describe a novel BDI execution framework that models context conditions as *decision trees*, rather than boolean formulae, allowing agents to *learn* the probability of success for plans based on experience. By using a probabilistic plan selection function, the agents can balance exploration and exploitation of their plans. We extend earlier work to include both parameterised goals and recursion and modify our previous approach to decision tree confidence to include large and even non-finite domains that arise from such consideration. Our evaluation on a pre-existing program that relies heavily on recursion and parametrised goals confirms previous results that naive learning fails in some circumstances, and demonstrates that the improved approach learns relatively well.

Keywords:
BDI Intelligent Agents, Machine Learning, Hybrid Systems

1. Introduction

Agents are an important technology that have the potential to take over contemporary methods for analysing, designing, and implementing complex software systems suitable for domains such as telecommunications, industrial control, business process management, transportation, logistics, and aeronautics [1, 2, 3, 4]. The BDI model of agency [5, 6] is a popular and well-studied

Email address: `{firstname.lastname}@rmit.edu.au` (Dhirendra Singh & Sebastian Sardina & Lin Padgham)

approach with substantial theoretical and practical work. It has its roots in philosophy with Bratman's [7] theory of practical reasoning and Dennett's theory of intentional systems [8]. A recent industry study [9] analysing several applications claimed that the use of BDI (Belief-Desire-Intention) agent technology in complex business settings can improve overall project productivity by up to 500%. Also the agent approach allowed the business to change and extend solutions quickly helping to bridge the semantic gap between the business side and IT development. BDI systems have built into them an ability to balance pro-actively pursuing a goal, with reactively responding to the environment. They also have a well developed failure recovery mechanism. This makes them very suitable for robotics applications operating in a physical world which is often more error prone than a software domain.

BDI systems, despite their strengths, do not however incorporate any ability to learn from experience. Our work makes a start at addressing this issue, focussing specifically on learning which plan to select next, to resolve a particular goal in a particular world state.

There are many agent programming languages and development platforms in the BDI tradition, including JACK [10], JADEX [11], and Jason [12] among others. All of them follow a similar basic architecture, whereby *abstract plans* written by programmers are combined and used *reactively* 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). A *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 then relies 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. Although this is quite flexible and effective, an important drawback is the lack of ability to learn from ongoing experience. An ability to *learn* plan selection in particular situations, adds a whole new layer of robustness and flexibility. Firstly there may be situations where it is difficult to determine in advance the exact situation under which a particular approach is likely to succeed. This is especially the case when it involves complex combinations of values of environmental variables. Secondly, an environment may change over time, or be slightly different in different deployment locations. The ability for the agent system to observe and learn from its performance is obviously a very desirable property.

In our work we achieve this by replacing (or augmenting) the usual boolean formula for representation of context conditions, by a decision tree [13] which is learnt based on experience. Our plan selection is then based on a probabilistic approach, usually choosing the plan which has the highest likelihood of success, based on experience. This probabilistic approach is also more suitable than the standard boolean approach for complex and often partially observable worlds, where various plans may be worth trying, but have different chances of success.

There are however various nuances that must be addressed for such *online* learning. There is the usual balance between exploration and exploitation evident in all learning. Moreover, learning is impacted by the structure imposed by the hierarchical representation of BDI programs. We have addressed these issues in previous papers [14, 15], looking at various approaches to the problem. In this paper we add the ability to deal with paramaterised goals and with recursive calls, both of which are essential for real applications. Unfortunately, once we add this expressivity our previous preferred approach does not scale. Consequently we develop a simplified approximation to achieve the same basic intuition which we have previously shown to be correct in principle. We then empirically evaluate our approach by taking an existing BDI program from the JACK tutorial, removing the context conditions, and learning the appropriate use of the plans provided using our framework.

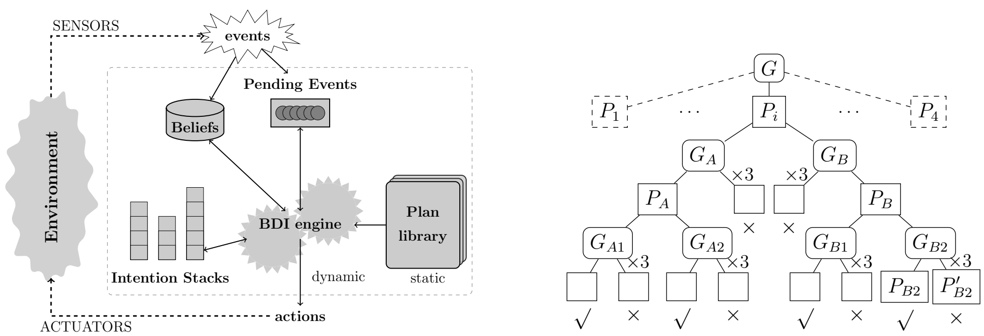
Our approach can easily be combined with the standard plan selection mechanism, by allowing the agent programmer to provide initial context conditions that could later be automatically "refined" by the agent system. By doing so, one can effectively take a BDI program and "tune it" using our learning framework. For simplicity, though, context conditions are learnt from scratch in our experimental work.

In the next section we introduce both BDI programming and our learning framework, as well as an overview of our previous approaches. We then describe in detail the learning framework that incorporates the additional aspects of parameterised goals and recursive calls, with our revised approach to address previously identified issues. We show empirical evaluation on an example program developed by Agent Oriented Software for their JACK tutorial, by removing the context conditions and applying our learning approach. We finish with a discussion of outstanding issues and related work.

2. Preliminaries

2.1. BDI Agent Systems

BDI agent-oriented programming is a popular, well-studied, and practical paradigm for building intelligent agents situated in complex and dynamic en-



(a) BDI-style architecture.

(b) A BDI Goal-Plan Hierarchy.

Figure 1: BDI Architecture and Goal-Plan Hierarchy.

vironments with (soft) real-time reasoning and control requirements [16, 9]. Generally speaking, BDI agent-oriented programming languages are built around an explicit representation of propositional attitudes (e.g., beliefs, desires, intentions, etc.). A BDI architecture addresses how these components are represented, updated, and processed to determine the agent's actions.

A BDI agent consists, basically, of a belief base (akin to a database) which stores the agent's knowledge about the world, a set of pending event-goals,[1] which include both external percepts or messages and internal goals, a plan library, and an intention structure. Figure 1(a) depicts a typical BDI architecture. The plan library contains rules of the form $e : \psi \leftarrow \delta$ indicating that δ is a suitable procedure for achieving event-goal e when context condition ψ is true. Among other operations, the plan body procedure δ will typically include the execution of actions (act) in the environment and subgoals ($!e$) that are in turn resolved by selecting suitable plans for those subgoals. For example, the following plan rules may be part of the plan library of an elevator controller:

$$\begin{aligned} \mathit{Serve}(floor) :&\quad \mathit{Serving}(floor) \leftarrow !\mathit{GoTo}(floor);\ \mathit{Open}; \mathit{Close}; \mathit{Off}(floor) \\ \mathit{GoTo}(floor) :&\quad \mathit{At}(x) \wedge x > floor \leftarrow \mathit{GoUp};\ !\mathit{GoTo}(floor) \end{aligned}$$

That is, to serve a request from a floor (i.e., event $\mathit{Serve}(floor)$) which the elevator is supposed to serve (i.e., condition $\mathit{Serving}(floor)$ is true), can be

[1] In this paper the terms *event-goal*, *event*, and *goal* are used interchangeably.

achieved by first going to the floor (by solving the $!GoTo(floor)$ sub-goal), then opening and closing the door, and finally turning the floor's request light off. In turn, to go to a floor that is above the current location of the elevator, it needs to go up one floor (i.e., execute primitive action $GoUp$) and then post again the (sub)goal of reaching the floor in question.

The basic *reactive goal-oriented behavior* of BDI systems involves the system responding to events by selecting an appropriate plan from the library, and placing its program body into the intention base, a structure containing the current, partially instantiated, plans that the agent has committed to in order to achieve some event-goals. A plan is appropriate if it is designed for the event in question (relevant) and its context condition is believed true (applicable). In contrast with traditional planning, execution happens at each step. The use of plans' context-preconditions to make choices as late as possible, together with the built-in goal-failure mechanisms, ensures that the system is responsive to changes in the environment. In this paper we focus on the plan library to investigate ways of learning appropriate or better plan selection based on experience.

By grouping together plans responding to the same event (or goal) type, the plan library can be seen as a set of *goal-plan tree* templates: a goal node has children representing the relevant plans for achieving it; and a plan node, in turn, has children representing the subgoals (including primitive actions) of the plan. These structures can be seen as AND/OR trees: for a plan to succeed all 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 hierarachical structure shown in Figure 1(b). 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, in other states, such plans may behave differently). 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 two working leaf 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 the low-level goal G_{B2}.

As can be seen, *plan-selection* is critically important. Standard BDI systems leverage domain expertise by means of the context conditions of plans. In this work, we are interested in exploring how a situated agent may *learn* plan selection based on experience, in order to improve goal achievement.

2.2. Learning for BDI Plan Selection

In order to facilitate learning regarding which plan should be executed for a given goal in a particular world state, we first replace each plan's boolean formula that is the standard representation for context conditions in BDI programming languages, with a decision tree [13] that provides a judgement as to whether the plan is likely to succeed or fail for the given situation.

To select plans based on information in the decision trees, we use a probabilistic method that chooses a plan based on its believed likelihood of success in the given situation. This approach provides a balance between exploitation (we choose plans with relatively higher success expectations more often), and exploration (we sometimes choose plans with lower success expectation to get better confidence in their believed applicability by trying them in more situations). This balance is important because ongoing learning influences future plan selection, and subsequently whether a good solution is learnt.

The resolution of goals in BDI execution results in the invocation of plans that in turn may post sub-goals that are further handled by sub-plans in a hierarchical manner. In a programming context, this is equivalent to making a function call that in turn calls sub-functions. However a sub-goal may have a number of possible plans for achieving it, some of which will work better in particular situations than others. In our learning context, where we do not yet know which plans work well in which situations, a plan may fail not because the plan was a bad choice in the given situation, but instead because the run-time choice of sub-plans was incorrect for the situation.

Our first approach [14] to address the learning problem was that of careful consideration whereby failures are recorded for learning purposes only when we are sufficiently sure that the failure was not due to poor sub-plan choices. We have shown that this conservative approach is more robust, though often slower, than a more aggressive approach which records all experiences, but can in some particular cases completely fail to learn.

Our second approach reported in [15] was to adjust the plan selection probability based on some measure of our confidence in the decision tree. We consider the reliability of a plan's decision tree in a given world state to be proportional to the number of sub-plan choices (or paths below the plan in the goal-plan hierarchy) that have been covered in that world state. Here *coverage* [15] refers to the set of explored paths relative to the set of all possible paths. The greater the coverage, the more we have explored and the greater the confidence in the resulting decision tree. By biasing the plan selection probability with a coverage-based confidence measure we achieved the same robustness as that of conservative recording of failure cases. The coverage approach, however, is more flexible as the extent to which this is used can be readily adjusted by parameters in the selection formula.

A limitation with the previous approaches is that events were assumed to be propositional atoms, i.e., parameterised event-goals were not considered. By *parameterised* we mean an event-goal that may contain "data" as part of its definition. For instance, event $travelTo(dest)$ may represent the goal to travel to location $dest$. In general, a goal to move to location A may require different strategies than those for addressing a goal to move to location B, and the learning must account for this. Another limitation is the assumption that the agent's plan library does not include recursive subgoaling, so that the goal-plan tree structure induced is always *finite*. For example, the above plan rule in the elevator controller's library for handling subgoal $GoTo(floor)$ would not be allowed, since its procedure involves posting the same subgoal event as the rule's head. Clearly both limitations would preclude the applicability of the approach in many practical domains where hierarchies are usually expressed in a compact manner by using parameterised goal events and plans, and often make use of (direct or indirect) recursive procedures to encode iterative strategies.

Furthermore, the coverage approach [15] does not scale to recursive structures. Conceptually we can unfold the recursive structure to a specified depth. However, the number of paths is exponential in the recursion number and further compounded by parameterised event-goals and the number of possible world states. An additional limitation is that coverage does not consider domain complexity. For instance, a leaf plan that has no sub-goals will achieve full coverage when it is tried once, after which selection will be fully biased towards the plan's decision tree classification. However, the decision tree that at this point has only witnessed one world will generalise the outcome to all as-yet-unseen worlds leading to misclassification.

In the following section we present the details of our learning approach, incorporating both parametrised goals, and recursion, as well as a new simpler confidence measure that is based on the general idea of coverage but does not suffer from its limitations.

3. The BDI Learning Framework

Our learning task is as follows: *Given past execution data and the current world state, determine which plan to execute next in order to best address the given event-goal.* In the BDI sense, our task is to learn the context condition of each plan in the goal-plan hierarchy. In this section we describe our BDI Learning Framework that enables such learning. In particular we describe the use of decision trees for learning context conditions and the confidence-based probabilistic plan selection that incorporates this learning, while focusing on parameterised event-goals and recursion.

3.1. Integrating Decision Trees into Context Conditions for Plans

A plans context condition is a logical formula that is constructed at design time and evaluated against an event-goal at run time to determine if the plan is applicable in the given world state.[2] To allow the context condition to be learnt over time, we annotate each plan's context formula with a *decision tree*.[3][14] The idea is that the agent starts with some *necessary but possibly insufficient* conditions for each plan (provided by the designer), and over time and in the course of trying plans in various world states will *refine* each plan's context condition using the learnt decision tree.

The choice of decision trees for learning is motivated by several factors. Firstly, decision trees support hypotheses that are a disjunction of conjunctive terms and this representation is compatible with how context formulas are generally written. Secondly, decision trees can be converted to *if-then* rules that are human readable and can be validated by a domain expert. Finally, decision trees are robust against training data that may contain errors. This is specially relevant in stochastic domains where applicable plans may nevertheless fail due to unforeseen circumstances.

For each plan, the training set for its decision tree contains samples of the form $[w, o]$, where w is the world state in which the plan was executed and o was the boolean outcome (success or failure). The world state w itself is a set of discrete attributes that together represent the state of affairs. Initially the training set is empty and grows as the agent tries the plan in various world states and records each result. Over time the decision tree learnt from

[2]Context formulas may reference internal beliefs as well as environment states, and for this study we treat both as included in the world state.

[3]It is perfectly feasible to combine the existing logical formula with the decision tree classification, but to aid our understanding of the decision tree learning in this study we always use an empty initial formula.

the training set will contain only those attributes of world state w that are relevant to that plans context condition.

The number of attributes in world state w and their range has a bearing on the size of the training set required to correctly learn the context condition. In general, world state w should be constructed with all attributes that are possibly relevant to the context condition. For instance, for a plan to pick objects using a robotic arm, the attributes *objectSurface* and *gripperWet* are likely relevant and should be included, while the attribute *dayOfWeek* possibly is not and may be exluded. The choice of attributes to include in the world state w is eventually a design decision and dependent on domain knowledge. For our purposes we assume that the designer provides a set of all attributes that are considered *possibly relevant* to the context condition of the plan.[4] In the worst case, this set is the full set of available attributes.

The decision tree inductive bias is a preference for smaller trees. In other words, the induction will trade-off some accuracy in classification for compactness of representation. This means that some training samples get incorrectly classified in the wrong "bucket" (where the bucket name is *success* or *failure* in our case) when the actual outcome class of those training samples is different. So in a given bucket (or class) one may get a total of m training samples out of which n are misclassified. The ratio $1 - (n/m)$ gives the likelihood of class membership,[5] and is the fraction we use as the expected likelihood of success of the plan.

3.2. Support for Recursive Event-Goals

Recursion in our context refers to the case where the resolution of an event-goal instance $G(\vec{x_1})$ involves first the resolution of goal-event instance $G(\vec{x_2})$ of the same type G. The result is a growing stack of pending $G(\vec{x_i})$ event-goals that eventually terminate in $G(\vec{x_n})$ whose parameters satisfy the termination conditions where a non-recursive plan choice is made.

In order to understand the impact of recursion on context learning, we use the notion of an *execution trace* of the form $G_0(\vec{x_0})[P_0 : w_0] \cdot G_1(\vec{x_1})[P_1 : w_1] \cdot \ldots \cdot G_n(\vec{x_1})[P_n : w_n]$, that represents a sequence of event-goals along with the plans selected to handle them and the world state in which the selections

[4]An automatic compilation of potential relevant propositions can be done by analysing, if available, the preconditions and effects of actions that might be executed when handling a goal. This, however, is out of the scope of this paper.

[5]In our study we use algorithm J48, a version of c4.5 [13], from the weka learning package [17] that automatically provides this ratio.

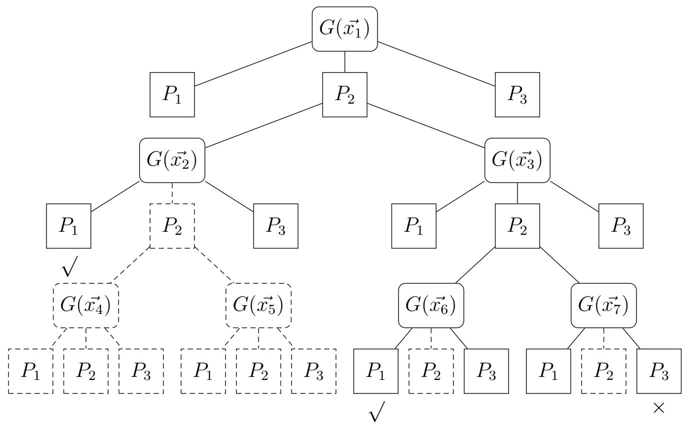


Figure 2: Goal-plan hierarchy containing a parameterised goal G handled by three plans P_1, P_2 and P_3. Here plan P_2 posts two instances of G resulting in recursion. Two levels of recursive unfolding are shown. Dashed P_2 nodes indicate unexplored recursive sub-trees.

were made. So $G_i(\vec{x_i})[P_i : w_i]$ captures the case where plan P_i was selected in world state w_i in order to achieve the goal-event $G_i(\vec{x_i})$.

Consider the example BDI goal-plan hierarchy of Figure 2. The structure has just a single parameterised goal G and three options to handle it, one of which, P_2, in turn posts two instances of the same parameterised goal G. In this way, the only plans that take an action in the environment are P_1 and P_3. The figure highlights an execution trace as follows:

$$\lambda = G(\vec{x_1})[P_2 : w_1] \cdot G(\vec{x_2})[P_1 : w_1] \cdot G(\vec{x_3})[P_2 : w_2] \cdot G(\vec{x_6})[P_1 : w_2] \cdot G(\vec{x_7})[P_3 : w_3].$$

The first choice in the execution results in the selection of plan P_2 to handle event-goal instance $G(\vec{x_1})$ in a given world w_1. Plan P_2 in turn immediately posts the event-goal instance $G(\vec{x_2})$ that is successfully handled by the non-recursive node P_1. Plan P_2 then posts the second event-goal instance $G(\vec{x_3})$, which then is handled by itself in a recursive manner. The outcome is that λ traces a path that involves the successive execution of leaf plan P_1 for event-goal $G(\vec{x_2})$ followed by another execution of P_1 for event-goal $G(\vec{x_6})$, finally terminating in the failure of leaf plan P_3 for event-goal $G(\vec{x_7})$. Note that if plan P_2 had instead been selected to handle $G(\vec{x_7})$ then a deeper recursive call would have ensued. Similarly if earlier in the execution trace plan P_2 was selected to handle event-goal $G(\vec{x_2})$ then a different recursive sub-tree (shown in Figure 2 as dotted nodes under $G(\vec{x_2})$) would have unfolded.

The immediate implication of a recursive goal-plan structure is that the size of the hierarchy is no longer static but instead unfolds in a dynamic manner. The risk then is that since the conditions that terminate recursion are not ready at the start (we are trying to learn them), then the agent may get trapped in an infinite recursive loop during exploration. This has implications for any strategy that relies on the structure being finite. For instance, our conservative recording approach [14] and coverage-based confidence measure [15] both suffer from this problem. Incidentally, the simpler aggressive recording approach [14] is not impacted by recursion as it does not consider the goal-plan structure.

One way to resolve this issue is to treat all recursive goals simply as subtrees in a static structure and limit the recursive *unfolding* to a maximum allowed depth. In this study we use this bounded recursion approach for handling recursive structures. It follows then that wherever a recursive structure applies, a maximum recursion value must be supplied. This may not be an unrealistic requirement given that the domain expert will usually have some idea of how much recursion is sufficient for a given parameterised event-goal.

3.3. Calculating Confidence in the Decision Tree Classification

The typical use of decision trees lies in the *offline* induction from a complete training set. In that sense, the use of decision trees in our framework is unorthodox since the training set is built incrementally by recording samples after each new execution. This results in the training set being incomplete in the early stages of learning,[6] leading to misclassification. A confidence measure in the decision tree classification is therefore desirable to address this issue. Previously [15] we showed how the coverage of possible execution paths below the plan in the goal-plan hierarchy may be used to build such a measure. Here we propose a new confidence measure that builds on the idea but that does not suffer from its limitations (Section 2.2).

Our requirement for the confidence measure is that it be a monotonic function whose values transition from no confidence (0.0) to full confidence (1.0) based on experience. Specifically, the experiences we are interested in should constitute coverage of the plan complexity (number of sub-plan choices) and the domain complexity (number of world states in which the plan applies). Since an exact calculation of such coverage does not scale for all practical purposes then we are interested in an *approximate* coverage that is still representative of the state of affairs but is simpler to compute.

[6]Training data is incomplete in the sense that the agent has only collected a portion of the full data set required to learn the correct classification.

One way to achieve this is to use a monotonic decay function[7] (for instance $\epsilon_i = \epsilon_{i-i} * \delta$ where $\delta < 1.0$) but where the decay factor δ is tied to the complexity involved. This way, a plan that has a larger number of sub-plan choices will utilise a slower decay factor δ taking longer to reach full confidence $(1 - \epsilon)$ than another plan that has less choices to make. For goal-plan complexity this decay δ_{Pt} may be calculated offline by analysing the goal-plan hierarchy. In this work, we have calculated δ_{Pt} in terms of average breadth and depth of the structure, where depth is the maximum level of recursion in this case, to provide an approximation of the complexity of the structure.[8] A similar treatment is possible for domain complexity although the decay factor in this case cannot be pre-determined since the number of world states is not known upfront and is dependent on the domain. For domain complexity then, it may be reasonable to treat the decay factor δ_{Pd} as a parameter specified by the domain expert.

$$c_P = (1.0 - \epsilon_{Pt}) * (1.0 - \epsilon_{Pd}). \tag{1}$$

Equation 1 shows how the final confidence c_P is calculated for a given plan P. Here ϵ_{Pt} is the plans tree complexity decay while ϵ_{Pd} is the plans domain complexity decay. The actual updates to the decay values are performed each time the plan P is executed while the rate of decay is governed by the decay factors δ_{Pt} and δ_{Pd} accordingly.

3.4. Handling Parameterised Event-Goals

Our BDI learning framework account presented earlier [14, 15] did not account for *parameterised event-goals* but only for event-goal instances. In practical BDI systems, it is often the case that a single plan will handle all instances of a parameterised event-goal. Furthermore event-goal instance parameters will generally be included in the context logical formula.

Consider again the goal-plan structure in Figure 2 and the highlighted solution path terminating in the leaf plans indicated by the $\surd$ symbol. An important point here is that the indicated solution applies to the event-goal *instance* $G(\vec{x_1})$ and to that instance alone. For a different instance $G(\vec{y_1})$

[7]This technique is frequently applied in machine learning algorithms for balancing between exploration of choices and exploitation of learning.

[8]There are other ways of calculating δ_{Pt} (e.g., in [15] we have used an accurate calculation of the number of choices below plans; however this is not feasible anymore when goals and plans are schemas with possibly infinite instances); the main idea is to somehow measure the complexity of a hierarchical structure, and is the subject of ongoing work.

the solution path would likely be different (one way to visualise this in the Figure 2 is to think of it as an animation where the event-goal parameters and the placement of the $\surd$ symbols changes on each frame). This means that event-goal instance parameters must also be considered as input for a plan's decision tree in order to learn solutions per event-goal instance.

We include such an account by augmenting the training samples for the decision tree with the event-goal parameters. As such, the training set now contains samples of the form $[w, x, o]$ where the world state w is the initial set of all relevant attributes that represent the state of affairs, x is the set of all event-goal parameters, and o is the outcome class (success or failure). Incorporating the event-goal parameter set x in the training data is sufficient for learning with parameterised event-goals, and no fundamental change to the framework is required.

3.5. Calculating Plan Selection Weights based on Confidence

Typical BDI platforms offer several mechanisms for plan selection from a set of applicable plans, such as plan precedence and meta-level reasoning. However, since these mechanisms are pre-programmed and do not take into account the experience of the agent, we provide a new *probabilistic plan selection* function for this purpose.

For each plan, given its expectation of success (as determined by its decision tree learning) and a confidence measure in this expectation (based on coverage), we calculate a final *selection weight* that is indicative of the likelihood of the plan being selected for execution. Equation 2 shows how the plan selection weight $\Omega_P(w)$ is calculated for a given world state w.[9] Initially, the confidence c_P is zero and the weight takes the default value of 0.5. Over time, as the confidence improves towards the final value of 1.0, the selection weight approaches the value $\kappa_P(w)$ estimated by the plan's decision tree.

$$\Omega_P(w) = 0.5 + \left[c_P * (\kappa_P(w) - 0.5)\right]. \tag{2}$$

Given the set of applicable plans for resolving event-goal G in world state w then, our probabilistic plan selection mechanism chooses a plan P_i with a probability directly proportional to its selection weight $\Omega_{P_i}(w)$. Such selection ensures a balance between the *exploitation* of current know-how and the *exploration* of new choices that is necessary for online learning tasks.

[9]The formulation of the plan selection weight is described in [15].

4. A Case Example: The Hanoi Towers Robot

To evaluate our learning framework we consider an existing BDI program from the JACK agent platform distribution [10]. The example involves a robot playing the well-known Towers of Hanoi game where the goal is to stack discs of decreasing size onto a single pin. The rules of the game forbid discs to be moved onto smaller discs, however top discs may be moved onto discs of larger size across three pins. The problem is interesting for our purposes since the example solution makes use of parameterised event-goals and recursion. Furthermore, unlike our previous evaluations [14, 15] with synthetic plan libraries, here the evaluation criteria is clear: *does our learning framework achieve the performance of the existing system?*

The example solution consists of a *Player* agent that solves the game for any given legal initial configuration. The game solving strategy is encoded in plan *DiscStacker* that solves for one disc at a time starting from the largest and ending with the smallest onto a chosen pin. This in turn is achieved by posting event $Solve(d,p)$ for solving disc d onto pin p. There are four plans that are relevant for this purpose:

SolveRight This plan solves moving a disc to the pin it is already on. Since the goal is already true, the plan does *nothing*.

SolveTopMove This plan moves the disc d to the destination pin p if the disc is not already there and if the move is legal. The actual move is perfomed by the primitive action $move(p2,p)$, where $p2$ is the source pin of disc d.

SolveTop This plan solves for the case when the disc d may be legally lifted but cannot be legally placed at the destination because the top disc on the destination pin is smaller than d. In this case, the plan first moves all the discs in the destination pin that are smaller than disc d to the third (auxiliary) pin, and then re-posts the sub-goal to move d to pin p i.e. $Solve(d,p)$.

SolveMiddle This plan solves moving a disc from the *middle* of a stack. In this case, the plan first clears the source pin so that disc d becomes the new top of the pin. This is done by solving for sub-goal $Solve(d2,p2)$ where disc $d2$ is the disc currently on top of d and $p2$ is the auxiliary. Subsequently the plan re-posts the sub-goal of moving d to pin p i.e. event $Solve(d,p)$.

Figure 3 illustrates the goal-plan hierarchy for the domain. Here we focus on learning the recursive parameterised $Solve(d,p)$ event for which we remove the context conditions from the example plans and apply our framework.

4.1. Experimental Setup

The aim of this study is to evaluate our learning framework for recursive event-goals. For this reason our experimentation with the Hanoi problem

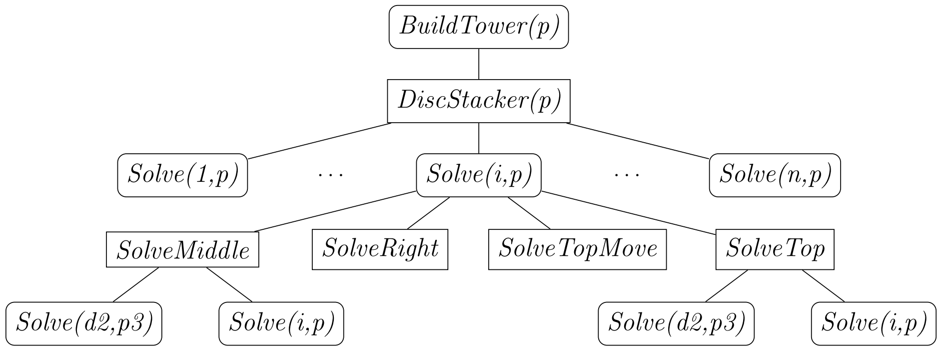


Figure 3: Goal-plan hierarchy for the Towers of Hanoi domain.

focusses on learning to resolve the recursive event *Solve* only and not on learning the strategy that solves the full Hanoi towers problem (this is done by *DiscStacker(p)*). Since the full set of possible *Solve* events and initial pin configurations is large, our first step is to construct a sufficiently rich subset that we will use to evaluate our learning approaches. nd then re-po We proceed by running the original Hanoi program for a number of randomly generated *Solve* events. For each run we record the *Solve* event, the initial pins configuration, and the maximum recursion encountered for the solution. This gives us a bag of several initial configurations for each recursion level that is a subset of all possible configurations.

Next, we run each candidate approach on the set of saved configurations for a given recursion level. i.e. where all solutions lie exactly at the specified recursion number. We use a fixed random generation seed for each experiment so that the same sequence of *Solve* events is generated for each learning approach. This isolates any environmental factors and allows us to attribute any differences in performance to the learning approaches alone.

4.2. Results

The following results are for a Hanoi problem with $five$ discs.[10] Each plans domain complexity decay factor for the confidence calculation of Equation 1 are set to $\delta_{Pd} = 0.9$. For goal-plan tree complexity, we use $\delta_{Pt} = 1$ for non-recursive plans and $\delta_{Pt} = \left[1 - (1/r^k)\right]$ for recursive plans. Here r is

[10]We use five discs in order to keep the state space rich enough yet sufficiently small to allow learning runs to be completed and evaluated in reasonable time.

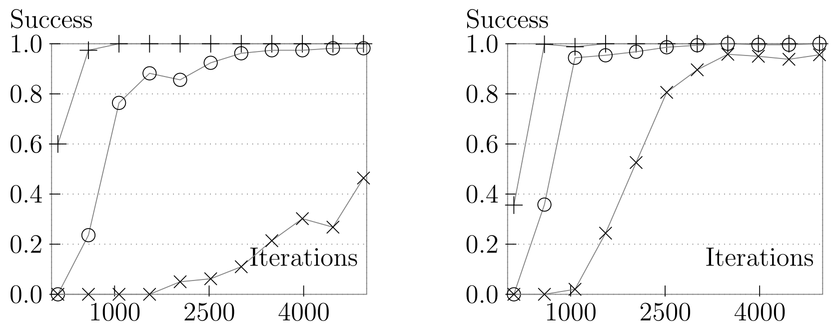
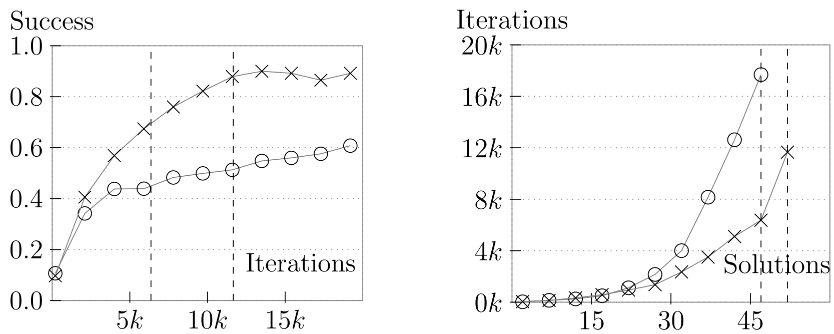


Figure 4: Agent performance under ACL and ACL+Ω schemes for solutions at recursion levels one (pluses), three (circles) and five (crosses). Each point represents results from 5 experiment runs using an averaging window of 100 samples.

the run-time recursion level and k is arbitrarily set to 4.0,[11] and controls the rate of change of decay i.e. the number of steps to reach full confidence. In all experiments, the recursion is bound to a maximum of *eight* levels that is sufficient to solve all configurations for a five-disc Hanoi problem. The performance of two learning configurations is contrasted. The baseline learning algorithm ACL refers to the original aggressive learning approach of [14] and [15] using the original probabilistic plan selection function that has no confidence-based bias (uses decision tree expectation of success only). The new algorithm is referred to as ACL+Ω and uses the same aggressive learning approach as the former but combined with the new confidence-based probabilistic selection function (Equation 2) presented in this study.

Experiment 1. To understand how the two approaches perform for solutions of varying difficulty we conducted a set of tests with solutions at different recursive levels. Each test consisted of resolving a known set of *Solve* event configurations, saved earlier as described in Section 4.1, whose solutions all required a given recursive depth. Figure 4 shows that as the solution difficulty increases from one to five recursion levels, ACL performance drops much more significantly compared to that of ACL+Ω. For instance, for solutions requiring five levels of recursion (crosses in Figure 4), ACL achieves only 50% success at $5k$ iterations whereas ACL+Ω achieves 95% success by $3.5k$ iterations. The poor performance of ACL may be attributed to the fact that

[11] In future work we hope to establish principles for determining general parameters.



(a) Success Rate (b) Solutions Found

Figure 5: Agent performance under ACL (circles) and ACL+Ω (crosses) schemes. Each point represents an average result from 5 experiment runs.

deeper solutions require more *move*(*p2*,*p*) steps and where an earlier success does not exist to guide selection at each resulting state the exploration is mostly random. On the other hand, the confidence-based measure of Equation 2 takes into account the goal-plan tree complexity and is able to guide the ACL+Ω exploration towards the deeper solutions.

Experiment 2. Next, we conducted an experiment that consisted of resolving the full set of saved *Solve* events i.e. the set of all solutions for recursion levels one to five. Figure 5(a) shows the results for the two approaches where ACL+Ω performs better than ACL as expected from the previous experiment results. For the same experiment, we also recorded the number of solutions found. Figure 5(b) shows that ACL+Ω resolves all 52 goals within $12k$ iterations whereas ACL resolves only 47 by the end of the experiment at $20k$. At a similar point of comparison, to resolve 47 goals ACL+Ω takes around $6.4k$ iterations whereas ACL takes more than twice as long at $17.7k$. The vertical dashed lines in Figure 5(a) and Figure 5(b) mark the 47th and 52nd solutions.

Experiment 3. Finally, we ran a third experiment to understand the impact of applicability thresholds. In the classical BDI framework plans are either applicable or not in a boolean decision. However, in our modified framework plans are applicable according to the selection weight given by Equation 2. Since plan execution is often not cost-free in real systems, it is likely that an adequate plan selection scheme would not select *any* plan if they all have too low an expectation of success, and an agent may fail a plan without even trying. To represent this scenario we setup an applicability threshold of 20% whereby plans with expectations of success less than this threshold would not be selected. In this case ACL shows a complete inability to learn as reported earlier [15]. In contrast ACL+Ω benefits from the adjusted selection weights of Equation 2 and shows similar performance as before (Figure 5(a)).[12]

Analysis. Observe in Figure 5(a) that ACL+Ω does not reach the performance of the hand-crafted JACK program and converges to about 90% success even though it successfully discovers all solutions (Figure 5(b)). This is because the decision tree representation does not guarantee that the training data will always be correctly classified (we discuss this accuracy versus compactness trade-off in Section 3.1). For instance, a decision tree may report a poor likelihood of success for a given state even when the associated training sample indicates success, due to the sample being misclassified in the failure "bucket". One could guarantee correctness by referencing the training data directly, for instance using a look-up table. However, the decision tree representation is preferable for its compact representation combined with the ability to generalise to as-yet-unseen world states. Currently, when our learnt decision trees are converted to rules they do not "look" like the original ones but are far more complex. This is mainly due to representational differences as our simple representation is propositional whereas the original conditions are relational.[13]

5. Discussion and Conclusion

This paper builds on earlier work [14, 15] that extends the typical BDI programming framework to use decision trees as (part of) a plan's context condition, with a probabilistic plan selection mechanism that caters for both exploration and exploitation of plans. Previously we have shown that due to the structure of BDI programs, care must be taken in how learning is used, to avoid problems in certain situations. In some cases these problems lead to failure to learn at all, as we also show here.

In this paper we extend previous work to allow for parameterised goals (e.g. $travelTo(dest)$) and also for recursion, both of which are necessary for

[12] The threshold of 0.2 while somewhat arbitrary is consistent with that used earlier [15]. The difference between the default weight (0.5) and the threshold weight (0.2 in this case) decides how much "give" we have in the exploration. The closer the threshold is to 0.5 the greater the chance that the plan will be aborted before a solution is found.

[13] We hope to address this using relational decision trees in future work.

real applications. In doing this our previous confidence measure which relied on a finite goal-plan tree did not scale, so we have provided an approximate measure, relying on the principles that have been shown correct, but without the limitations. This paper also takes an existing BDI program involving parameterised goals and recursion, and evaluates our approach using this program. By removing the existing context conditions, and then learning the correct behaviour, we show that we are able to obtain good (although not perfect)[14] performance. We also demonstrate that the naive approach to learning, that does not account for the BDI program structure fails to learn given some program structures and an applicability threshold.

There is still work to be done before our framework may be applied for *practical* on-line learning in situated agents. Firstly, the framework has not been integrated with standard BDI failure handling and recovery. Clearly this will be needed (and is the subject of ongoing work), but we do not expect this to undermine any results described here. In fact a careful integration of failure handling could improve the speed of learning as multiple attempts could be made to achieve a (sub)-goal. However care needs to be taken regarding changes to world state and possible interactions between failed attempts and eventually successful ones.

Secondly, our use of decision trees is naive. For instance, currently execution data is maintained forever and decision trees re-built after each plan execution. Furthermore, we learn using actual world states, whereas an improvement would be to learn using relational world information. While not ideal, our setup nonetheless allows us to focus on the nuances of learning in BDI programs first without worrying about the underlying techniques.

Finally, we do not detect and learn interactions between sibling goals in the context of a particular parent; each subgoal is treated "locally." To handle such interactions, the selection of a plan for resolving a sub-goal should also be predicated on the goals higher than the sub-goal, that is, it should take into account the "reasons" for the sub-goal. Addressing this would require substantial modification to the BDI programming style in terms of representation, which is out of the scope of this work.

The issue of combining learning and deliberative approaches for online decision making in BDI-like systems has not been widely addressed. Hernández et al. [18] give a preliminary account of how decision trees may be induced on plan failures in order to find alternative logical context conditions in a

[14] We would hope that when learning is combined with programmer provided context conditions, the problems preventing perfect learning here would be avoided.

deterministic paint-world example. In [19] learnt user preferences are incorporated during BDI plan selection in a dialogue manager application using a decision tree learner. In contrast, [20] take the approach of refining existing BDI plans or learning new plans as a sequence of recorded actions based on prescriptions provided by the domain expert. In [21], low level robot soccer skills are learnt offline and then used in the deliberative decision making process once deployed. More recently, [22] give a comprehensive account of integrating learning in BDI deliberation for a real world ship berthing logistics domain. Here a neural network module is first trained offline on the available shipping port data and then used in a deployed BDI system to improve plan selection. Their results show significant improvement in berth productivity over the existing system of human operators.

A closely related area to BDI is that of hierarchical task network (HTN) systems where task decompositions used are similar to BDI goal-plan hierarchies. Recently, in similarly motivated work to ours, [23] proposed a method for learning 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. In contrast, in our work learning and deliberation are fully integrated in a way that one impacts the other and the classical exploration/exploitation dilemma applies.

The BDI architecture has also been shown [24] to be related to Markov Decision Processes that are heavily used for solving optimisation problems in reinforcement learning [13]. A sub-area of work related to ours is hierarchical reinforcement learning [25] where task hierarchies similar to BDI are used. When the aim is to find locally optimal solutions for each sub-MDP in the hierarchy, similar issues as ours arise, such as goal inter-dependence. In general, global optimality is possible only when information is fed into the sub-task (i.e. value functions use the entire state space), consistent with our analysis of goal inter-dependence issues. Interestingly, work by Dietterich [26] also supports the use of simultaneous learning at all levels (similar to our ACL based approaches) instead of waiting for the children to converge (analogous to our conservative approach [15]).

Although there is still work to do before we can expect learning to be successfully integrated into a fully autonomous BDI agent, the work reported here is significant in that it provides a solid foundation for adding new capabilities to BDI agents to allow them to learn and adapt based on experience.

[1] N. R. Jennings, An agent-based approch for building complex software systems, Communications of the ACM 44 (4) (2001) 35–41.

[2] R. A. Belecheanu, S. Munroe, M. Luck, T. Payne, T. Miller, P. McBurney, M. Pechoucek, Commercial applications of agents: Lessons, experiences and challenges, in: Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS), ACM Press, Hakodate, Japan, 2006, pp. 1549–1555.

[3] M. Ljungberg, A. Lucas, The OASIS air-traffic management system, in: Proceedings of the Pacific Rim International Conference on Artificial Intelligence (PRICAI), Seoul, Korea, 1992.

[4] Z. Tu, M. Ferry, G. Prickett, C. Heinze, Truly autonomous UAVs and teaming, in: Twelfth Australian Aeronautical Conference, 2007, pp. 483–489, engineers Australia.

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

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

[7] M. Bratman, Intentions, Plans, and Practical Reason, Harvard University Press, 1987.

[8] D. C. Dennet, The intentional Stance, MIT Press, 1987.

[9] S. S. Benfield, J. Hendrickson, D. Galanti, Making a strong business case for multiagent technology, in: Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS), ACM Press, 2006, pp. 10–15.

[10] P. Busetta, R. Rönnquist, A. Hodgson, A. Lucas, JACK intelligent agents: Components for intelligent agents in Java, AgentLink Newsletter 2 (1999) 2–5, Agent Oriented Software.

[11] A. Pokahr, L. Braubach, W. Lamersdorf, JADEX: Implementing a BDI-infrastructure for JADE agents, EXP - in search of innovation (Special Issue on JADE) 3 (3) (2003) 76–85.

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

[13] T. Mitchell, Machine Learning, McGraw Hill, 1997.

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

[15] D. Singh, S. Sardina, L. Padgham, S. Airiau, Learning context conditions for BDI plan selection, in: Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS), Toronto, Canada, 2010, to appear.

[16] M. P. Georgeff, F. F. Ingrand, Decision making in an embedded reasoning system, in: Proceedings of IJCAI, Detroit, USA, 1989, pp. 972–978.

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

[18] A. Guerra-Hernández, A. E. Fallah-Seghrouchni, H. Soldano, Learning in BDI Multi-agent Systems, Vol. 3259 of LNCS, Springer, 2004, pp. 218–233.

[19] A. Nguyen, W. Wobcke, An adaptive plan-based dialogue agent: integrating learning into a BDI architecture, in: Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS), ACM Press, New York, NY, USA, 2006, pp. 786–788. doi:10.1145/1160633.1160771.

[20] S. Karim, B. Subagdja, L. Sonenberg, Plans as Products of Learning, IEEE Computer Society, Washington, DC, USA, 2006, pp. 139–145. doi:10.1109/IAT.2006.100.

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

[22] P. Lokuge, D. Alahakoon, Improving the adaptability in automated vessel scheduling in container ports using intelligent software agents, European Journal of Operational Research 177 (3) (2007) 1985–2015.

[23] H. Zhuo, D. Hu, C. Hogg, Q. Yang, 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.

[24] G. I. Simari, S. Parsons, On the relationship between MDPs and the BDI architecture, in: Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS), ACM Press, New York, NY, USA, 2006, pp. 1041–1048. doi:10.1145/1160633.1160818.

[25] A. Barto, S. Mahadevan, Recent Advances in Hierarchical Reinforcement Learning, Discrete Event Dynamic Systems 13 (4) (2003) 341–379.

[26] T. Dietterich, Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition, Journal of Artificial Intelligence Research 13 (2000) 227–303.

Dhirendra Singh is a PhD candidate at the School of Computer Science & Information Technology at RMIT University, Melbourne, Australia, with co-supervision from the The Commonwealth Scientific and Industrial Research Organisation (CSIRO). His research focus is in the area of practical learning for BDI agent systems.

Prior to commencing his PhD in 2007, Dhirendra was developing high end simulators for Digital Signal Processors at Freescale Semiconductor (formerly part of Motorola). During his time there, he was primarily focussed on the development of Full Chip Simulators for the StarCore family of DSPs.

Dhirendra has been involved in AI research since his undergraduate days. He was a member of the RMIT RoboCup team that competed in the Middle-sized robot league in 1999. During that time he developed a software simulator to help speed agent development by allowing robot behaviours to be tested independent of the hardware. After graduation, Dhirendra continued his work in software agents as a researcher with The Centre for Scientific and Technological Research (ITC-IRST) where he helped build a proof-of-concept software prototype integrating software agents, web services, and distributed data sources.

Dhirendra completed his Bachelor of Engineering (Computer Systems) and a Bachelor of Applied Science (Computer Science) degrees from RMIT University in 2000.

Sebastian Sardina is a Research Fellow in the Intelligent Agents group at RMIT University, Melbourne, Australia. Sebastian completed his PhD at the Cognitive Robotics Group, University of Toronto, Canada, under the supervision of Hector Levesque. Prior to that, he obtained his Bachelor of Science from the Universidad Nacional del Sur in Bahia Blanca, Argentina.

Sebastian's research interests include reasoning about action and change; automated planning & planning within agent programming languages; agent ability and knowledge precondition for agent programs; formal models of deliberation/planning with sensing; procedural & declarative specification of goals; and logic programming for agents.

Sebastian has acted on the Program Committe of several events. In 2009, he was on the Program Committee of GENPLAN (within ICAPS), AI Australia, and Commonsense, and on the Senior Program Committee for IJCAI.

Lin Padgham is Professor of Artificial Intelligence in the School of Computer Science and I.T. at RMIT University, Melbourne, Australia. She has a PhD from University of Linkoping, Sweden, 1989. Lin's research interests are in various aspects of commonsense reasoning with an emphasis on formal methods for knowledge representation which are coupled with computationally realisable algorithms.

Lin's current research is in the area of intelligent multi-agent systems, focussing on modeling, building and understanding intelligent agents for complex application areas requiring a balance between goal directed long-term behaviour and reactive response to a dynamic environment. She has investigated various extensions to standard 'Belief Desire Intention' (BDI) reasoning, including planning, goal conflicts, and learning.

Lin has been Program Chair for AAMAS 2008, is on the Editorial Board of the Journal for Autonomous Multi Agent Systems, and regularly serves on the Program Committee for major international conferences such as KR, IJCAI and ECAI.