Available online at www.sciencedirect.com

ScienceDirect

Cognitive Systems Research 43 (2017) 1–20

Cognitive Systems RESEARCH

www.elsevier.com/locate/cogsys

A hybrid POMDP-BDI agent architecture with online stochastic planning and plan caching

Action editor: Rajiv Khosla

Gavin Rens [a,c,*], Deshendran Moodley [b,c]

[a] School of Mathematics, Statistics and Computer Science, University of KwaZulu-Natal, South Africa
[b] Department of Computer Science, University of Cape Town, South Africa
[c] Centre for Artificial Intelligence Research, CSIR Meraka, South Africa

Received 22 May 2016; received in revised form 15 October 2016; accepted 5 December 2016
Available online 21 December 2016

Abstract

This article presents an agent architecture for controlling an autonomous agent in stochastic, noisy environments. The architecture combines the partially observable Markov decision process (POMDP) model with the belief-desire-intention (BDI) framework. The Hybrid POMDP-BDI agent architecture takes the best features from the two approaches, that is, the online generation of reward-maximizing courses of action from POMDP theory, and sophisticated multiple goal management from BDI theory. We introduce the advances made since the introduction of the basic architecture, including (i) the ability to pursue and manage multiple goals simultaneously and (ii) a plan library for storing pre-written plans and for storing recently generated plans for future reuse. A version of the architecture is implemented and is evaluated in a simulated environment. The results of the experiments show that the improved hybrid architecture outperforms the standard POMDP architecture and the previous basic hybrid architecture for both processing speed and effectiveness of the agent in reaching its goals.

Keywords: Autonomous agents; POMDP; BDI; Satisfaction; Planning; Memory

1. Introduction

Imagine a scenario where a planetary rover has five tasks of varying importance. The tasks could be, for instance, collecting gas (for industrial use) from a natural vent at the base of a hill, taking a temperature measurement at the top of the hill, performing self-diagnostics and repairs, reloading its batteries at the solar charging station and collect soil samples wherever the rover is. The rover is programmed to know the relative importance of collecting soil samples. The rover also has a model of the probabilities with which its various actuators fail and the probabilistic noise-profile of its various sensors. The rover must be able to reason (plan) in real-time to pursue the right task at the right time while considering its resources and dealing with various events, all while considering the uncertainties about its actions (actuators) and perceptions (sensors).

We propose an architecture for the proper control of an agent in a complex environment such as the scenario described above. The architecture combines belief-desire-intention (BDI) theory (Bratman, 1987; Rao & Georgeff, 1995) and partially observable Markov decision processes (POMDPs) (Lovejoy, 1991; Monahan, 1982). Traditional BDI architectures (BDIAs) cannot deal with probabilistic uncertainties and they do not generate plans in real-time.

* Corresponding author at: Department of Computer Science, University of Cape Town, South Africa.
E-mail address: gavinrens@gmail.com (G. Rens).

http://dx.doi.org/10.1016/j.cogsys.2016.12.002
1389-0417/© 2016 Elsevier B.V. All rights reserved.

A traditional POMDP cannot manage goals (major and minor tasks) as well as BDIAs can. Next, we analyse the POMDPs and BDIAs in a little more detail.

One of the benefits of agents based on BDI theory, is that they need not generate plans from scratch; their plans are already (partially) compiled, and they can act quickly once a goal is focused on. Furthermore, the BDI framework can deal with multiple goals. However, their plans are usually not optimal, and it may be difficult to find a plan which is applicable to the current situation. That is, the agent may not have a plan in its library which exactly 'matches' what it ideally wants to achieve. On the other hand, POMDPs can generate optimal policies on the spot to be highly applicable to the current situation. Moreover, policies account for stochastic actions in partially observable environments. Unfortunately, generating optimal POMDP policies is usually intractable. One solution to the intractability of POMDP policy generation is to employ a *continuous planning* strategy, or *agent-centred search* (Koenig, 2001). Aligned with agent-centred search is the *forward-search* approach or *online* planning approach in POMDPs (Ross, Pineau, Paquet, & Chaib-draa, 2008).

The traditional BDIA maintains goals as *desires*; there is no reward for performing some action in some state. The reward function provided by POMDP theory is useful for modeling certain kinds of behavior or preferences. For instance, an agent based on a POMDP may want to avoid moist areas to prevent its parts becoming rusty. Moreover, a POMDP agent can generate plans which can optimally avoid moist areas. But one would not say that avoiding moist areas is the agent's *task*. And POMDP theory maintains a single reward function; there is no possibility of weighing alternative reward functions and pursuing one at a time for a fixed period—all objectives must be considered simultaneously, in one reward function. Reasoning about objectives in POMDP theory is not as sophisticated as in BDI theory. A BDI agent cannot, however, simultaneously avoid moist areas *and* collect gas; it has to switch between the two or combine the desire to avoid moist areas with every other goal.

The Hybrid POMDP-BDI agent architecture (or HPB architecture, for short) has recently been introduced (Rens & Meyer, 2015). It combines the advantages of POMDP theoretic reasoning and the potentially sophisticated means-ends reasoning of BDI theory in a coherent agent architecture. In this paper, we generalize the management of goals by allowing for each goal to be pursued with different intensities, yet concurrently.

Typically, BDI agents do not deal with stochastic uncertainty. Integrating POMDP notions into a BDIA addresses this. For instance, an HPB agent will maintain a (subjective) belief state representing its probabilistic (uncertain) belief about its current state. Planning with models of stochastic actions and perceptions is possible in the HPB architecture. The tight integration of POMDPs and BDIAs is novel to this architecture, especially in combination with desires with changing intensity levels.

This article serves to introduce two significant extensions to the first iteration (Rens & Meyer, 2015) of the HPB architecture. The first extension allows for multiple intentions to be pursued simultaneously, instead of one at a time. In the previous architecture, only one intention was actively pursued at any moment. In the new version, one agent action can take an agent closer to more than one goal at the moment the action is performed – the result of a new approach to planning. As a consequence of allowing multiple intentions, the policy generation module (Section 4.3), the desire function and the method of focusing on intentions (Section 4.2) had to be adapted. The second extension is the addition of a *plan library*. Previously, a policy (conditional plan) would have to be generated periodically and regularly to supply the agent with the recommendations of actions it needs to take. Although one of the strengths of traditional BDI theory is the availability of a plan library with pre-written plans for quick use, a plan library was excluded from the HPB architecture so as to simplify the architecture's introduction. Now we propose a framework where an agent designer can store hand-written policies in a library of plans and where generated policies are stored for later reuse. Every policy in the library is stored together with a 'context' in which it will be applicable and the set of intentions which it is meant to satisfy. There are two advantages of introducing a plan library: (i) policies can be tailored by experts to achieve specific goals in particular contexts, giving the agent *immediate* access to recommended courses of action in those situations and (ii) providing a means for policies, once generated, to be stored for later reuse so that the agent can take advantage of past 'experience' – saving time and computation.

In Section 2, we review the necessary theory, including POMDP and BDI theory. In Section 3, we describe the basic HPB architecture. The extensions to the basic architecture are presented in Section 4. Section 5 describes simulation experiments in one domain in which the proposed architecture is extensively tested, evaluating the performance on various dimensions. The results of the experiments confirm that the approach may be useful in some domains. In the last two sections, we discuss related work, points out some future directions for research in this area, and draw final conclusions.

2. Preliminaries

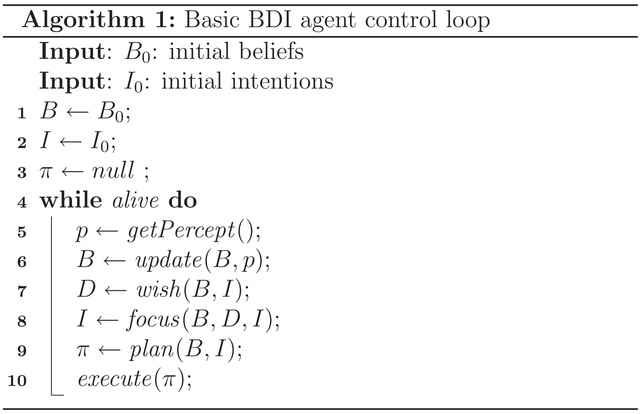
The basic components of a BDI architecture (Wooldridge, 1999, chap. 1; Wooldridge, 2002) are

- a set or knowledge-base B of beliefs;
- an option generation function *wish*, generating the objectives the agent would ideally like to pursue (its desires);
- a set of desires D (goals to be achieved);
- a 'focus' function which selects intentions from the set of desires;

- a structure of intentions I of the most desirable options/desires returned by the focus function;
- a library of plans and subplans;
- a 'reconsideration' function which decides whether to call the focus function;
- an execution procedure, which affects the world according to the plan associated with the intention;
- a sensing or perception procedure, which gathers information about the state of the environment; and
- a belief update function, which updates the agent's beliefs according to its latest observations and actions.

Exactly how these components are implemented result in a particular BDI architecture.

Algorithm 1. Basic BDI agent control loop



Algorithm 1: Basic BDI agent control loop

```
Input: $B_0$: initial beliefs
Input: $I_0$: initial intentions
$B \leftarrow B_0$;
$I \leftarrow I_0$;
$\pi \leftarrow null$ ;
while $alive$ do
    $p \leftarrow getPercept()$;
    $B \leftarrow update(B, p)$;
    $D \leftarrow wish(B, I)$;
    $I \leftarrow focus(B, D, I)$;
    $\pi \leftarrow plan(B, I)$;
    $execute(\pi)$;
```

Algorithm 1 (adapted from Wooldridge (2000, Fig. 2.3)) is a basic BDI agent control loop. π is the current plan to be executed. $getPercept(\cdot)$ senses the environment and returns a percept (processed sensor data) which is an input to $update(\cdot)$, which updates the agent's beliefs. $wish : B \times I \rightarrow D$ generates a set of desires, given the agent's beliefs, current intentions and possibly its innate motives. It is usually impractical for an agent to pursue the achievement of all its desires. It must thus filter out the most valuable and achievable desires. This is the function of $focus : B \times D \times I \rightarrow I$, taking beliefs, desires and current intentions as parameters. Together, the processes performed by *wish* and *focus* may be called deliberation, formally encapsulated by the *deliberate* procedure. $plan(\cdot)$ returns a plan from the plan library to achieve the agent's current intentions.

A more sophisticated controller would have the agent consider *whether* to re-deliberate, with a *reconsider* function placed just before deliberation would take place. The agent could also test at every iteration through the main loop whether the currently pursued intention is still possibly achievable. Serendipity could also be taken advantage of by periodically testing whether the intention has been achieved, without the plan being fully executed. Such an agent is considered 'reactive' because it executes one action per loop iteration; this allows for deliberation between executions. There are various mechanisms which an agent might use to decide when to reconsider its intentions. See, for instance, Bratman (1987), Pollack and Ringuette (1990), Kinny and Georgeff (1991), Kinny and Georgeff (1992), Schut and Wooldridge (2000), Schut and Wooldridge (2001), Schut, Wooldridge, and Parsons (2004).

In a partially observable Markov decision process (POMDP), the actions the agent performs have non-deterministic effects in the sense that the agent can only predict with a likelihood in which state it will end up after performing an action. Furthermore, its perception is noisy. That is, when the agent uses its sensors to determine in which state it is, it will have a probability distribution over a set of possible states to reflect its conviction for being in each state.

Formally (Kaelbling, Littman, & Cassandra, 1998), a POMDP is a tuple $\langle S, A, T, R, Z, P, b^0 \rangle$ with

- S, a finite set of states of the world (that the agent can be in),
- A a finite set of actions (that the agent can choose to execute),
- a transition function $T(s, a, s')$, the probability of being in s' after performing action a in state s,
- $R(a, s)$, the immediate reward gained for executing action a while in state s,
- Z, a finite set of observations the agent can perceive in its world,
- a perception function $P(s', a, z)$, the probability of observing z in state s' resulting from performing action a in some other state, and
- b^0 the initial probability distribution over all states in S.

In general, we regard an observation as the signal recognized by a sensor; the signal is generated by some event which is not directly perceivable.

A belief state b is a set of pairs $\langle s, p \rangle$ where each state s in b is associated with a probability p. All probabilities must sum up to one, hence, b forms a probability distribution over the set S of all states. To update the agent's beliefs about the world, a special function $SE(z, a, b) = b_n$ is defined as

$$b_n(s') = \frac{P(s', a, z) \sum_{s \in S} T(s, a, s') b(s)}{Pr(z|a, b)}, \tag{1}$$

where a is an action performed in 'current' belief state b, z is the resultant observation and $b_n(s')$ denotes the probability of the agent being in state s' in 'new' belief state b_n. Note that $Pr(z \mid a, b)$ is a normalizing constant.

Let the *planning horizon h* (also called the *look-ahead depth*) be the number of future steps the agent plans ahead each time it plans. $V^*(b, h)$ is the *optimal* value of future courses of actions the agent can take with respect to a finite horizon h starting in belief state b. This function assumes that at each step the action that will maximize the state's value will be selected.

Because the reward function $R(a,s)$ provides feedback about the utility of a particular state s (due to a executed in it), an agent who does not know in which state it is in cannot use this reward function directly. The agent must consider, for each state s, the probability $b(s)$ of being in s, according to its current belief state b. Hence, the *believed* reward for a in b is $\sum_{s\in S}R(a,s)b(s)$.

The optimal *state-value* function is define by

$$V^*(b,h) := \max_{a\in\mathscr{A}}\left[\sum_{s\in S}R(a,s)b(s) + \gamma\sum_{z\in Z}Pr(z\,|\,a,b)V^*(SE(z,a,b),h-1)\right],$$

where $0 \leqslant \gamma \leqslant 1$ is a factor to discount the value of future rewards and $Pr(z\,|\,a,b)$ denotes the probability of reaching belief state $b_n = SE(z,a,b)$. While V^* denotes the optimal value of a belief state, function Q^* denotes the optimal *action-value*:

$$Q^*(a,b,h) := \sum_{s\in S}R(a,s)b(s) + \gamma\sum_{z\in Z}Pr(z\,|\,a,b)V^*(SE(z,a,b),h-1)$$

is the value of executing a in the current belief state, plus the total expected value of belief states reached thereafter.

3. The basic HPB architecture

In BDI theory, one of the big challenges is to know *when* the agent should switch its current goal and *what* its new goal should be (Schut et al., 2004). To address this challenge, we propose that an agent should maintain intensity levels of desire for every goal. This intensity of desire could be interpreted as a kind of emotion. The goals most intensely desired should be the goals sought (the agent's intentions). We also define the notion of how much an intention is satisfied in the agent's current belief state. For instance, suppose that out of five possible goals, the agent currently most desires to watch a film and to eat a snack. Then these two goals become the agent's *intentions*. However, eating is not allowed inside the film-theatre, and if the agent were to go buy a snack it would miss the beginning of the film. So the total reward for first watching the film then buying and eating a snack is higher than first eating then watching. As soon as the film-watching goal is satisfied, it is no longer an intention. But while the agent was watching the film, the desire-level of the (non-intention) goal of being at home has been increasing. However, it cannot become an intention because snack-eating has not yet been satisfied. Going home cannot simply become an intention and dominate snack-eating, because the architecture is designed so that current intentions have precedence over non-intention goals, else there is a danger that the agent will vacillate between which goals to pursue. Nonetheless, snack-eating may be ejected from the set of intentions under the special condition that the agent is having an unusually hard time achieving it. For instance, if someone stole its wallet in the theatre, the agent can no longer have the current intention (i.e., actively pursue) eating a snack. Hence, in our architecture, if an intention takes 'too long' to satisfy, it is removed from the set of intentions. As soon as the agent gets home or is close to home, the snack-eating goal will probably become an intention again and the agent will start making plans to satisfy eating a snack. Moreover, the desire-level of snack-eating will now be very high (it has been steadily increasing) and the agent's actions will be biased towards satisfying this intention over other current intentions (e.g., over getting home, if it is not yet there).

A Hybrid POMDP-BDI (HPB) agent (Rens & Meyer, 2015) maintains (i) a belief state which is periodically updated, (ii) a mapping from goals to numbers representing the level of desire to achieve the goals, and (iii) the current set of intentions, the goals with the highest desire levels (roughly speaking). As the agent acts, its desire levels are updated and it may consider choosing new intentions and discard others based on new desire levels. Refer to Fig. 1 for an overview of the operational semantics. The figure refers to concepts defined in the following subsection.

3.1. Declarative semantics

The *state* of an HPB agent is defined by the tuple $\langle B,D,I\rangle$, where B is the agent's current belief state (i.e., a probability distribution over the states S, defined below), D is the agent's current desire function and I is the agent's current intention. More will be said about D and I a little later.

An HPB agent could be defined by the tuple $\langle Atrb,G,A,Z,T,P,Util\rangle$, where

- *Atrb* is a set of attribute-sort pairs (for short, the *attribute set*). For every $(atrb : sort) \in Atrb, atrb$ is the name or identifier of an attribute of interest in the domain of interest, like `BatryLevel` or `Direction`, and *sort* is the set from which *atrb* can take a value, for instance, real numbers in the range $[0,55]$ or a list of values like $\{\texttt{North},\texttt{East},\texttt{West},\texttt{South}\}$. So $\{(\texttt{BatryLevel} : [0,55]),(\texttt{Direction} : \{\texttt{North}, \texttt{East}, \texttt{West}, \texttt{South}\})\}$ could be an attribute set.

A state s is induced from *Atrb* as one possible way of assigning values to attributes: $s = \{(atrb : v)|(atrb : sort) \in Atrb, v \in sort,$ if $(atrb : v),(atrb' : v') \in s$ and $atrb = atrb'$, then $v = v'\}$. The set of all possible states is denoted S.

- G is a set of goals. A goal is a subset of some state $s \in S$. For instance, $\{(\texttt{BatryLevel} : 13),(\texttt{Direction} : \texttt{South})\}$ is a goal, and so are $\{(\texttt{BatryLevel} : 33)\}$

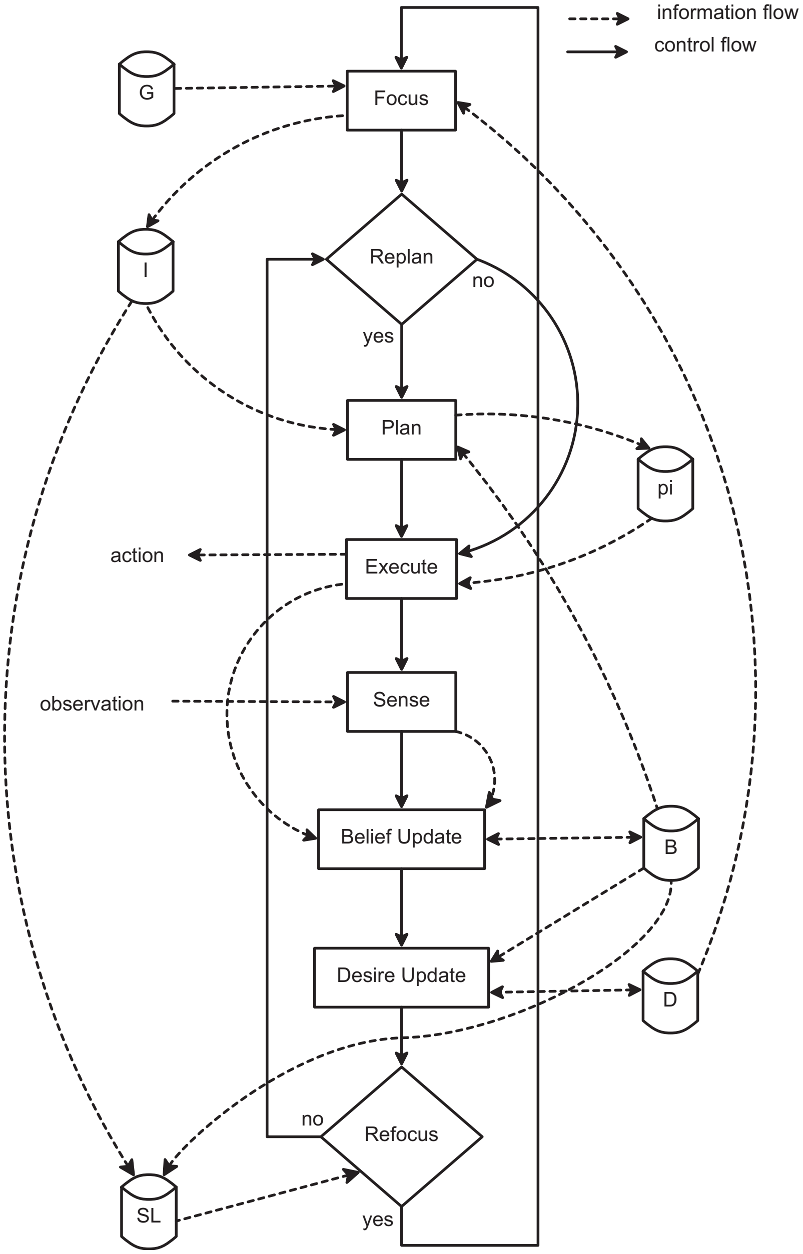


Fig. 1. Operational semantics of the basic HPB architecture. SL stands for *Satf_levels*. Note that *Satf_levels* depends on the current belief state and not on desire levels. Planning is also independent of desire levels. The focus function depends on desire levels and on satisfaction levels. In the case of plans consisting of a single action, the Replan decision node always returns 'yes'.

and $\{(\mathtt{Direction} : \mathtt{West})\}$. The set of goals is given by the agent designer as 'instructions' about the agent's tasks.

- A is a finite set of actions.
- Z is a finite set of observations.
- T is the transition function of POMDPs.
- P is the perception function of POMDPs.
- *Util* consists of two functions *Pref* and *Satf* which allow an agent to determine the utilities of alternative sequences of actions. $Util = \langle Pref, Satf \rangle$.*Pref* is the preference function with a range in $\mathbb{R} \cap [0, 1]$. It takes an action a and a state s, and returns the preference (any real number) for performing a in s. That is, $Pref(a, s) \in [0, 1]$. Numbers closer to 1 imply greater preference and numbers closer to 0 imply less preference. Except for the range restriction of $[0, 1]$, it has the same definition as a POMDP reward function, but its name indicates that it models the agent's preferences and not what is typically thought of as rewards. An HPB agent gets 'rewarded' by achieving its goals. The preference function is especially important to model action costs; the agent should prefer 'inexpensive' actions. *Pref* has a local flavor. Designing the preference function to have a value lying in [0,1] may sometimes be

challenging, but we believe it is always possible.*Satf* is the satisfaction function with a range in $\mathbb{R} \cap [0,1]$. It takes a state s and an intention I, and returns a value representing the degree to which the state satisfies the intention. That is, $Satf(I,s) \in [0,1]$. It is completely up to the agent designer to decide how the satisfaction function is defined, as long as numbers closer to 1 mean more satisfaction and numbers closer to 0 mean less satisfaction. *Satf* has a global flavor.

Fig. 1 shows a flow diagram representing the operational semantics of the basic HPB architecture.

3.2. The desire function

The desire function D is a total function from goals in G into the positive real numbers $\mathbb{R}^+$. The real number represents the intensity or level of desire of the goal. For instance, $(\{(BatryLevel: 13), (WeekDay: Tue)\}, 2.2)$ could be in D, meaning that the goal of having the battery level at 13 and the week-day Tuesday is desired with a level of 2.2. $(\{(BatryLevel: 33)\}, 56)$ and $(\{(WeekDay: Wed)\}, 444)$ are also examples of desires in D.

I is the agent's current intention; an element of G; the goal with the highest desire level. This goal will be actively pursued by the agent, shifting the importance of the other goals to the background. The fact that only one intention is maintained makes the HPB agent architecture quite different to standard BDIAs.

We propose the following desire update rule.

$$D(g) \leftarrow D(g) + 1 - Satf_\beta(g,B) \qquad (2)$$

Rule 2 is defined so that as $Satf_\beta(g,B)$ tends to one (total satisfaction), the intensity with which the incumbent goal is desired does not increase. On the other hand, as $Satf_\beta(g,B)$ becomes smaller (more dissatisfaction), the goal's intensity is incremented. The rule transforms D with respect to B and g. A goal's intensity should drop the more it is being satisfied. The update rule thus defines how a goal's intensity changes over time with respect to satisfaction.

Note that desire levels never decrease. This does not reflect reality. It is however convenient to represent the intensity of desires like this: only *relative* differences in desire levels matter in our approach and we want to avoid unnecessarily complicating the architecture.

3.3. Focusing and satisfaction levels

Focus is a function which returns one member of G called the (current) intention I. In the initial version of the architecture, the goal selected is the one with the highest desire level. After every execution of an action in the real-world, *Refocus* is called to decide whether to call *Focus* to select a new intention. *Refocus* is a meta-reasoning function analogous to the *reconsider* function mentioned in Section 2. It is important to keep the agent focused on one goal long enough to give it a reasonable chance of achieving it. It is the job of *Refocus* to recognize when the current intention seems impossible or too expensive to achieve.

Let *Satf_levels* be the sequence of satisfaction levels of the current intention since it became active and let *MRY* be a designer-specified number representing the length of a subsequence of *Satf_levels*—the *MRY* last satisfaction levels.

One possible definition of *Refocus* is

$$Refocus(c,\theta) \overset{def}{=} \begin{cases} \text{'no'} & \text{if } |Satf\_levels| < MRY \\ \text{'yes'} & \text{if } c < \theta \\ \text{'no'} & \text{otherwise,} \end{cases}$$

where c is the average change from one satisfaction level to the next in the agent's 'memory' *MRY*, and θ is some threshold, for instance, 0.05. If the agent is expected to increase its satisfaction by at least, say, 0.1 on average for the current intention, then θ should be set to 0.1. With this approach, if the agent 'gets stuck' trying to achieve its current intention, it will not blindly keep on trying to achieve it, but will start pursuing another goal (with the highest desire level). Some experimentation will likely be necessary for the agent designer to determine a good value for θ in the application domain.

Note that if an intention was not well satisfied, its desire level still increases at a relatively high rate. So whenever the agent focuses again, a goal not well satisfied in the past will be a top contender to become the intention (again).

3.4. Planning for the next action

A basic HPB agent controls its behavior according to the policies it generates. *Plan* is a procedure which generates a POMDP policy π of depth h. Essentially, we want to consider all action sequences of length h and the belief states in which the agent would find itself if it followed the sequences. Then we want to choose the sequence (or at least its first action) which yields the least cost and which ends in the belief state most satisfying with respect to the intention.

Planning occurs over an agents belief states. The satisfaction and preference functions thus need to be defined for belief states: The satisfaction an agent gets for an intention in its current belief state is defined as

$$Satf_\beta(I,B) := \sum_{s\in S} Satf(I,s)B(s),$$

where $Satf(I,s)$ is defined above and $B(s)$ is the probability of being in state s. The definition of $Pref_\beta$ has the same form as the reward function ρ over belief states in POMDP theory:

$$Pref_\beta(a,B) := \sum_{s\in S} Pref(a,s)B(s),$$

where $Pref(a,s)$ was discussed above.

During planning, preferences and intention satisfaction must be maximized. The main function used in the *Plan* procedure is the HPB action-state value function Q^{*}_{HPB}, giving the value of some action a, conditioned on the current belief state B, intention I and look-ahead depth h:

$$\begin{aligned} Q^{*}_{HPB}(a,B,I,h) &:= \alpha Satf_{\beta}(I,B) + (1-\alpha)Pref_{\beta}(a,B) \\ &\quad + \gamma\sum_{z\in Z} Pr(z|a,B)\max_{a'\in A} Q^{*}_{HPB}(a',B',I,h-1), \\ Q^{*}_{HPB}(a,B,I,1) &:= \alpha Satf_{\beta}(I,B) + (1-\alpha)Pref_{\beta}(a,B), \end{aligned}$$

where $B' = SE(a,z,B), 0 \leqslant \alpha \leq 1$ is the goal/preference 'trade-off' factor, γ is the normal POMDP discount factor and SE is the normal POMDP state estimation function.

Plan returns $\arg\max_{a\in A} Q^{*}_{HPB}(a,B,I,h)$, the trivial policy of a single action.

4. The extended HPB architecture

The operational semantics of the extended architecture is similar to the first version, except that each goal is now given its own weight of importance, and a plan library is now involved. The agent starts off with an initial set of intentions, a subset of its goals. For the current set of intentions, it must either select a plan from the plan library or generate a plan to pursue all its intentions. At every iteration of the agent's control loop, an action is performed, an observation is made, the belief state is updated, and a decision is made whether to modify the set of intentions. After several actions of the current policy (conditional plan) are executed, the agent seeks a new policy by consulting its plan library, and if an adequate policy is not found, generating one.

In the next subsection, we introduce some new notation and changes made to the architecture. Section 4.2 discusses how the focussing procedure must change to accommodate the changes. Section 4.3 explains how policies are generated for simultaneous pursuit of multiple goals. Finally, Section 4.4 presents the plan library, which was previously unavailable, and how the agent and agent designer can use it to their benefit.

4.1. Prologue

The HPB agent model gets a few new components – a goal weight function W, a compatibility function $Cpbl$, the plan library $Lbry$ and a set of thresholds $Tlds$. It can thus be defined by the tuple

$$\langle Atrb, G, W, Cpbl, A, Z, T, P, Util, Lbry, Tlds\rangle.$$

In the previous version, satisfaction and preference were traded-off by a "trade-off factor" which was not explicitly mentioned in the agent model. Actually the trade-off factor should have been part of the model, because it must be provided by the agent designer, and it directly affects the agent's behavior. In the new version, every goal $g \in G$ will be weighted by $W(g)$ according to the importance of g to the agent. Goal weights are constrained such that $W(g) > 0$ for all $g \in G$, and $\sum_{g\in G} W(g) = 1$.

A fundamental extension is that I becomes a *set* of intentions. In this way, an HPB agent may actively pursue several goals simultaneously. For example, a planetary rover may want to travel to its recharging station and simultaneously make same atmospheric measurements en route.

The first version has also been changed so that the set of goals G is simply a set of names, rather than restricting a goal to be a set of attribute values, as was previously done. Goals are defined by how they are used in the architecture, particularly by their involvement in the definition of satisfaction and reward functions.

In the extended architecture, *Util* is more expressive. It will also be convenient to use more compact notation. Here we let $Util = \langle \kappa, \rho, \sigma\rangle$, where κ is the same as $Pref$, σ is the same as $Satf$ and ρ is the reward function. To clarify:

- We move away from a *preference* function, and rather think of a *cost* function κ. Preferences will be captured by the set of satisfaction functions.
- $\rho : A \times G \times S \rightarrow [0,1]$ replaces the use of *Satf* in the value function (for planning), but ρ has access to actions, which makes reward specification easier. In other words, $\rho(a,g,s)$ is the reward for performing a in s towards achieving g.
- In the original version, *Satf* was used both in the value function and in tracking the agent's overall progress towards a goal. Now, *Satf* (aka σ) is used only for the latter. ρ cannot be used for satisfaction *tracking* because when overall goal satisfaction must be measured, there is no access to a particular action. In other words, while an agent thinks about its satisfaction for a goal *between* planning sessions, it does not think in terms of satisfaction with respect to particular actions.

As a consequence of being able to pursue several goals at the same time, there exists a danger that the agent will pursue one intention when it necessarily causes another intention to become less satisfied. For instance, visiting the USA regional headquarters is diametrically opposite to visiting the China regional headquarters at the same time. Other examples of goals which should be 'disjoint' are $\mathtt{work-in-garden}$ and $\mathtt{have-lunch}$, and $\mathtt{recharge-battery}$ and $\mathtt{replace-battery}$. The solution we use is to list, for each goal $g \in G$, all other goals which are *compatible* with it, in the sense that their simultaneous pursuit 'effective' (defined by the agent designer). Let $Cpbl(g)$ denote the set of goals compatible with g. It is mandatory that $g \in Cpbl(g)$. Two goals g and g' are called *incompatible* if and only if $g' \notin Cpbl(g)$ or $g \notin Cpbl(g')$.

Suppose $G = \{\mathtt{visit-USA-HQ}, \mathtt{visit-China-HQ}, \mathtt{work-in-garden}, \mathtt{have-lunch}, \mathtt{recharge-}$

$\texttt{battery}, \texttt{replace} - \texttt{battery}\}$. Then an agent designer may specify

$$Cpbl(\texttt{visit} - \texttt{USA} - \texttt{HQ}) = \{\texttt{visit} - \texttt{USA} - \texttt{HQ}, \texttt{have} - \texttt{lunch}\}$$

and

$$Cpbl(\texttt{recharge} - \texttt{battery}) = \{\texttt{recharge} - \texttt{battery}, \texttt{work} - \texttt{in} - \texttt{garden}, \texttt{have} - \texttt{lunch}\}.$$

Note that $\texttt{work} - \texttt{in} - \texttt{garden}$ and $\texttt{have} - \texttt{lunch}$ are incompatible.

Lbry is a set of plans, where each plan is a policy and a context in which that policy is recommended. The details are given in Section 4.4. As will be seen, periodically, an agent needs to check whether its current context matches one of these policy contexts to decide whether to adopt said policy. A 'match' depends on one or two designer-defined thresholds θ_i and θ_b. These two thresholds plus the threshold θ_f used in the intention focusing strategy (see Section 4.2) are specified in $Tlds := \{\theta_f, \theta_i, \theta_b\}$.

4.2. A new approach to focusing

Given that *I* is a *set* of intentions, ensuring that the 'correct' goals are intentions at the 'right' time to ensure that the agent behaves as desired, requires some careful thought. It is still important to keep the agent focused on one intention long enough to give it a reasonable chance of achieving it, temporarily stop pursuing intentions it is struggling to achieve.

The HPB architecture does not have a focus *function* which returns a subset of *G* of intentions *I*. Rather, we have a set of procedures which decide at each iteration which intention to remove from *I* (if any) and which goal to add to *I* (if any). Incompatibility must also be dealt with.

Let $Satf\_levels(g)$ be the sequence of satisfaction levels of some goal $g \in I$ since g became active (i.e., was added to *I*) and let *MRY* be a number representing the length of a sub-sequence of $Satf\_levels(g)$—the *MRY* last satisfaction levels of goal *g*. *Remove* is defined exactly like *Refocus*:

$$Remove(g, I) := \begin{cases} \text{'no'} & \text{if } |Satf\_levels(g)| < MRY(g) \\ \text{'yes'} & \text{if } \delta(g) < \theta_f \\ \text{'no'} & \text{otherwise,} \end{cases}$$

where $\delta(g)$ is the average change from one satisfaction level of g to the next in the agent's 'memory', and θ_f is the threshold above which $\delta(g)$ must be for g to remain an intention.

Let *MI* be the currently *most intense* goal defined as

$$MI := \arg\max_{g \in G} D(g).$$

We define two focusing strategies for *sets* of intentions: the over-optimistic strategy and the compatibility strategy.

4.2.1. Over-optimistic strategy

This strategy ignores compatibility issues between goals. In this sense, the agent is (over) optimistic that it can successfully simultaneously pursue goals which are incompatible.

Add *MI* to *I* only if $MI \notin I$. If *MI* is added to *I*, clear *MI*'s record of satisfaction levels, that is, let $Satf\_levels(MI)$ be the empty sequence.

Next: For every $g \in I$, if $|I| > 1$ and $Remove(g, I)$ returns 'yes', then remove *g* from *I*.

4.2.2. Compatibility strategy

Add *MI* to *I* only if $MI \notin I$ and there does not exists a $g \in I$ such that $g \notin Cpbl(MI)$. If *MI* is added to *I*, clear *MI*'s record of satisfaction levels, that is, let $Satf\_levels(MI)$ be the empty sequence.

Next: For every $g \in I$, if $|I| > 1$ and $Remove(g, I)$ returns 'yes', then remove *g* from *I*.

There is one case which must still be dealt with in the compatibility strategy: Suppose for some $g \in G, \bar{g} \notin Cpbl(g)$. Further suppose that $I = \{\bar{g}\}$ (i.e., $|I| = 1$) and g is and remains the most intensely desired goal. Now, g may not be added to *I* because it is incompatible with $\bar{g}$, no other goal will be attempted to be added to *I* and $\bar{g}$ may not be removed while it is the only intention, *even if* $Remove(\bar{g}, I)$ returns 'yes'. What could easily happen in this case is that g will continually increase in desire level, $\bar{g}$'s average satisfaction level will remain below the change threshold (i.e., $\delta(\bar{g}) < \theta_f$ remains true), and the agent continues to pursue only $\bar{g}$. To remedy this 'locked' situation, the following procedure is run after the previous 'add' and 'remove' procedures are attempted. If $I = \{\bar{g}\}, \bar{g} \notin Cpbl(MI)$ and $Remove(\bar{g}, I)$ returns 'yes', then remove $\bar{g}$ from *I*, add *MI* to *I* and clear *MI*'s record of satisfaction levels.

4.2.3. A new desire function

The old rule (in new notation) is still available:

$$D(g) \leftarrow D(g) + W(g)(1 - \sigma_\beta(g, B)), \tag{3}$$

where the satisfaction an agent gets for a goal g at its current belief state is defined as

$$\sigma_\beta(g, B) := \sum_{s \in S} \sigma(g, s) B(s),$$

where $\sigma(g, s)$ is defined above and $B(s)$ is the probability of being in state s.

We have found through experimentation that when an intention-goal's desire levels are updated, non-intention-goals may not get the opportunity to become intentions. In other words, it may happen that whenever new non-intention-goals are considered to become intentions, they are always 'dominated' by goals with higher levels of desire which are already intentions. By disallowing intentions' desire levels to increase, non-intentions get the opportunity to 'catch up' with their desire levels. A new form of the desire update rule is thus proposed for this version of the architecture:

$$D(g) \leftarrow D(g) + (1 - i(I,g))W(g)(1 - \sigma_\beta(g,B)) \tag{4}$$

The term $(1 - i(I,g))$ in (4) ensures that a goal's desire level changes if and only if the goal is not an intention.

Both rules (3) and (4) have the effect that as $\sigma_\beta(g,B)$ tends to one (total satisfaction), the intensity with which the incumbent goal is desired does not increase. On the other hand, as $\sigma_\beta(g,B)$ becomes smaller (more dissatisfaction), the goal's intensity is incremented—by at most its weight of importance $W(g)$. A goal's intensity should drop the more it is being satisfied.

However, update rule (3) which is independent of whether a goal is an intention may still result in better performance in particular domains. (This question needs more research.) It is thus left up to the agent designer to define which form of the rule better suits the application domain.

4.3. Planning by policy generation

In this section, we shall see how the planner can be extended to compute a policy which pursues several goals simultaneously. Goal weights are also incorporated into the action-state value function.

The main function used in the *Plan* procedure is the HPB action-state value function Q^*_{HPB}, giving the value of some action a, conditioned on the current belief state B and look-ahead depth h:

$$\begin{aligned} Q^*_{HPB}(a,B,I,h) :=\ & i(I,g_1)W(g_1)\rho_\beta(a,g_1,B) + \cdots \\ & + i(I,g_n)W(g_n)\rho_\beta(a,g_n,B) - \kappa_\beta(a,B) \\ & + \gamma\sum_{z\in Z} Pr(z|a,B)\max_{a'\in A} Q^*_{HPB}(a',B',I,h-1), \end{aligned}$$

$$\begin{aligned} Q^*_{HPB}(a,B,I,1) :=\ & i(I,g_1)W(g_1)\rho_\beta(a,g_1,B) + \cdots \\ & + i(I,g_n)W(g_n)\rho_\beta(a,g_n,B) - \kappa_\beta(a,B), \end{aligned}$$

where

- $i(I,g_j) = 1$ if $g_j \in I$, else $i(I,g_j) = 0$ if $g_j \notin I$,
- $W(g)$ is the weight for achieving g,
- $\langle g_1,\ldots,g_n\rangle$ is an ordering of the goals in G,
- $\rho_\beta(\cdot)$ and $\kappa_\beta(\cdot)$ are the expected (w.r.t. a belief state) values of $\rho(\cdot)$, resp., $\kappa(\cdot)$,
- $B' = SE(a,z,B)$,
- γ is the normal POMDP discount factor and
- SE is the normal POMDP state estimation function.

Now, instead of *Plan* returning a single action (assuming $h > 1$), *Plan* generates a tree-structures plan of depth h, conditioned on observations, that is, a *policy*. With a policy of depth h, an agent can execute a sequence of up to h actions, where the choice of exactly which action to take at each step depends on the observation received just prior. $\arg\max_{a\in A} Q^*_{HPB}(a,B,I,h)$ is used at every choice point to construct the policy.

Fig. 2 is a graphical example of a policy with two actions and two observations. The agent is assumed to be in belief state B^{cur} when the policy is generated. At every belief state node (triangles), the optimal action is recommended. After an action is performed, all/both observations are possible and thus considered. There is thus a choice at every $\circ$ node; however, it is not a choice for the agent, rather, it is a choice for the environment which observation to send to the agent. Given the action performed, for every possible observation, a different belief state is generated. At every $\triangleright$ node (belief state), $\arg\max_{a\in A} Q^*_{HPB}(a,\triangleright,I,h)$ is applied to

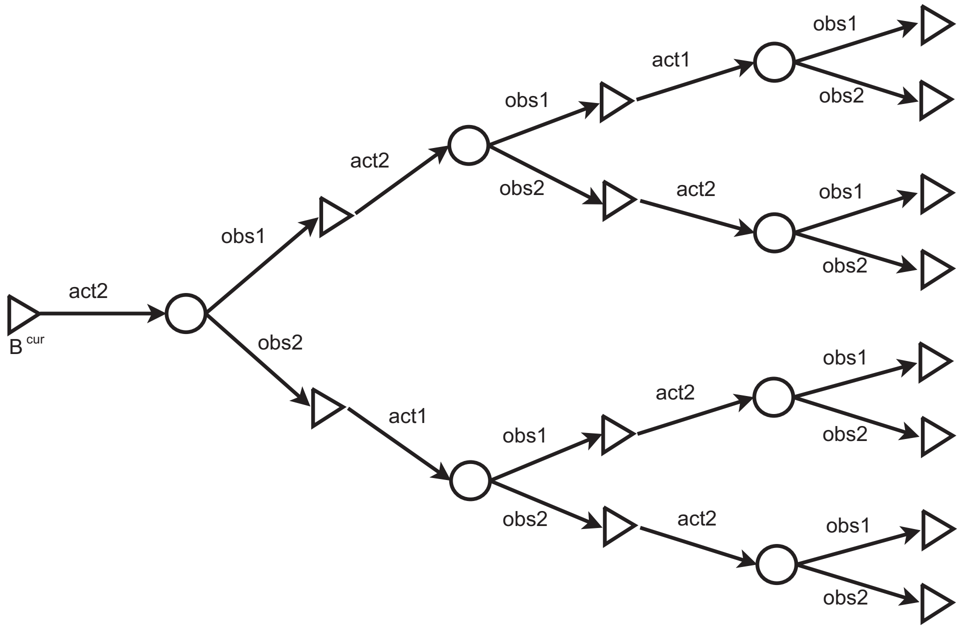


Fig. 2. An example policy of depth 3.

determine the action to perform there. (In theory, the agent can choose to perform any action at these $\triangleright$ nodes, but our agent will take the recommendations of POMDP theory for optimal behavior.) The agent will perform `act2` first, then depending on whether `obs1` or `obs2` in sensed, the agent should next (according to the policy) perform `act2` , respectively, `act1` . Then a third action will be performed according to the policy and conditional on which observation is sensed.

4.4. Introducing a plan library

Another extension of the basic architecture is that a language based on the attributes is introduced. The language L is the set of all *sentences*. Let ϕ and ψ be sentences. Then the following are also sentences.

- $\texttt{true}$,
- $(atrb : v)$, i.e., an attribute-value pair,
- $\phi \wedge \psi$,
- $\phi \vee \psi$,
- $\neg\phi$.

If a sentence ϕ is *satisfied* or *true* in a state s, we write $s \Vdash \phi$. The semantics of L is defined by

- $s \Vdash \texttt{true}$ always,
- $s \Vdash (atrb : v) \iff (atrb : v) \in s$,
- $s \Vdash \phi \wedge \psi \iff s \Vdash \phi$ and $s \Vdash \psi$,
- $s \Vdash \phi \vee \psi \iff s \Vdash \phi$ or $s \Vdash \psi$,
- $s \Vdash \neg\phi \iff$ not $s \Vdash \phi$.

Let Φ be a sentence in L. When a sentence in L appears in a written policy (see below), it is called a *context*.

We define two kinds of plans: an *attribute condition plan* is a triple $I : \Phi : \pi$, and a *belief state condition plan* is a triple $I : B : \pi$, where I is a set of intentions, π is a POMDP policy, Φ is a context and B is a belief state. All plans are stored in a *plan library*.

The idea is that attribute condition plans (abbreviation: a-plans) are written by agent designers and are available for use when the agent is deployed. Roughly speaking, belief state condition plans (abbreviation: b-plans) are automatically generated by a POMDP planner and stored when no a-plan is found which 'matches' the agent's current belief state and intention set.

4.4.1. Attribute condition plans

Policies in a-plans are of two kinds:

Definition 1 (*Most likely context*). An a-plan *most-likely-context* policy is either an action or has the form

$$a : \{(\Phi_1, \pi_1), \ldots, (\Phi_n, \pi_n)\},$$

where a is an action, the Φ_i are contexts, and each of the π_i is one of the two kinds of a-plan policies.

At belief state B, the *degree of belief* of Φ is

$$Degree(\Phi, B) := \sum_{s \in S, s \Vdash \Phi} B(s).$$

We abbreviate "most-likely-context" as 'ml'. If an ml policy $\pi = a : ML$ is adopted for execution and it is not simply an action, then a is executed, an observation is received, the current belief state is updated to B' and finally the policy which is paired with the most likely context is executed – that is,

$$\underset{\pi' : \quad (\Phi', \pi') \in ML}{\arg\max} \; Degree(\Phi', B')$$

is executed.

Definition 2 (*First applicable context*). An a-plan *first-applicable-context* policy is either an action or has the form

$$a : \langle(\Phi_1 \bowtie p_1, \pi_1), \ldots, (\Phi_n \bowtie p_n, \pi_n)\rangle,$$

where a is an action, the Φ_i are contexts, $\bowtie := \{\leqslant, \geqslant\}$, the p_i are probabilities, and each of the π_i is one of the two kinds of a-plan policies.

We abbreviate "first-applicable-context" as 'fa'. If an fa policy $\pi = a : FA$ is adopted for execution and it is not simply an action, then a is executed, an observation is received, the current belief state is updated to B' and finally the policy which is paired with the *first* context which satisfies its probability inequality is executed - that is, π_i is executed such that $Degree(\Phi_i, B') \bowtie p_i$ and $(\Phi_i \bowtie p_i, \pi_i) \in FA$ and there is no $(\Phi_j \bowtie p_j, \pi_j) \in FA$ such that $j < i$ for which $Degree(\Phi_j, B') \bowtie p_j$. If no context in the sequence $\langle(\Phi_1 \bowtie p_1, \pi_1), \ldots, (\Phi_n \bowtie p_n, \pi_n)\rangle$ satisfies its inequality, the a-plan of which the policy is a part is regarded as having finished, that is, the control loop is then in a position where a fresh plan in the plan library is sought.

In the following example a-plan policy, an agent must move around in a six-by-six grid world to collect items. Suppose the plan selected from the library is $I : \Phi : \pi$ with I being $\{\texttt{six} - \texttt{one}, \texttt{collect}\}$, Φ being

$$\begin{aligned}
&((\texttt{direction} : \texttt{North}) \vee (\texttt{direction} \\
&\qquad : \texttt{West})) \wedge \neg(\texttt{x} - \texttt{coord} \\
&\qquad : 6) \wedge \neg(\texttt{y} - \texttt{coord} : 1)
\end{aligned}$$

and π being

$$\begin{aligned}
&\texttt{move} - \texttt{forward} : \{ \\
&\quad ((\texttt{item} - \texttt{here} : \texttt{yes}), \texttt{take} - \texttt{item}), \\
&\quad ((\texttt{item} - \texttt{here} : \texttt{no}), \texttt{move} - \texttt{forward} : \langle \\
&\qquad ((\texttt{x} - \texttt{coord} : 6), (\texttt{direction} : \texttt{North}) \\
&\qquad \geqslant 0.9, \texttt{turn} - \texttt{left}), \\
&\qquad ((\texttt{y} - \texttt{coord} : 1), (\texttt{direction} : \texttt{West}) \\
&\qquad \geqslant 0.9, \texttt{turn} - \texttt{right}), \\
&\qquad (\texttt{true} \leqslant 1), \texttt{move} - \texttt{forward})\rangle\}
\end{aligned}$$

One can see that π itself is an ml policy, but embedded inside it is an fa policy.

Algorithm 2. *FindPolicy*

```
Algorithm 2: FindPolicy
  Input: $B^{cur}$: current belief state
  Input: $I^{cur}$: current intention set
  Input: $\theta_i$: intention-set threshold
  Input: $\theta_b$: belief state threshold
  Input: $PlanLib$: the plan library
  Input: $h$: planning horizon / policy depth
  Output: A POMDP policy of depth $h$
  $ApplicablePlans \leftarrow \{I^{lib} : \Phi : \pi^{lib} \in PlanLib \mid IS(I^{cur}, I^{lib}) \geq \theta_i\}$;
  if $ApplicablePlans \neq \emptyset$ then
      return $\arg\max_{\pi^{lib}:\ I^{lib}:\Phi:\pi^{lib} \in ApplicablePlans} Degree(\Phi, B^{cur})$;
  if $ApplicablePlans = \emptyset$ then
      $ApplicablePlans \leftarrow \{I^{lib} : B^{lib} : \pi^{lib} \in PlanLib \mid BS(I^{cur}, I^{lib}, B^{cur}, B^{lib}) \geq \theta_i\}$;
  if $ApplicablePlans \neq \emptyset$ then
      $ApplicablePlans \leftarrow \{I^{lib} : B^{lib} : \pi^{lib} \in ApplicablePlans \mid Sim_\beta(B^{lib}, B^{cur}) \geq \theta_b\}$;
  if $ApplicablePlans \neq \emptyset$ then
      return $\arg\max_{\pi^{lib}:\ I^{lib}:B^{lib}:\pi^{lib} \in ApplicablePlans} Sim_\beta(B^{lib}, B^{cur})$;
  if $ApplicablePlans = \emptyset$ then
      $\pi^{cur} \leftarrow Policy(B^{cur}, I^{cur}, h)$;
      Add $I^{cur} : B^{cur} : \pi^{cur}$ to $PlanLib$;
      return $\pi^{cur}$;
```

With respect to a-plans, whether two intention sets match will be determined by how many goals they have in common. Thus, the similarity between I^{cur} and I^{lib} can be determined as follows.

$$IS(I^{cur}, I^{lib}) := \frac{\sum_{g \in G, g \in I^{cur} \cap I^{lib}} 1}{|I^{cur} \cup I^{lib}|}.$$

$IS(\cdot)$ lies in $[0, 1]$. I^{cur} and I^{lib} need not have equal cardinality. Larger values of $IS(\cdot)$ mean more similarity/ closer match. The agent designer can decide what value of $IS(I, I')$ constitutes a 'match' between I and I' (see the discussion on "thresholds" below).

4.4.2. Belief state condition plans

Recall that b-plans have the form $I : B : \pi$, where I is a set of intentions, B is a belief state, and π is a POMDP policy. What constitutes a match between intention sets with respect to b-plans is different: Policies generated at two times t and t' might be significantly different for the same (similar) context(s) if the satisfaction levels of the intentions are significantly different at the two times. This is an important insight because policies of b-plans are *generated*, not written. Even though $IS(I^{cur}, I^{lib})$ may constitute a 'match' (with I^{lib} in a b-plan), π^{lib} might be completely impractical for pursuing I^{cur}. The measure of similarity will be the sum of differences between satisfaction levels. Note that an intention's satisfaction levels can only be compared if the intention appears in both intention sets under consideration. We denote the similarity between two intention sets I^{cur} and I^{lib} as $BS(I^{cur}, I^{lib}, B^{cur}, B^{lib})$ and define it as follows.

$$BS(I^{cur}, I^{lib}, B^{cur}, B^{lib}) := \frac{\sum_{g \in G, g \in I^{cur} \cap I^{lib}} 1 - ||\sigma_\beta(g, B^{cur}) - \sigma_\beta(g, B^{lib})||}{|I^{cur} \cup I^{lib}|}.$$

where $||x||$ denotes the absolute value of x. $BS(\cdot)$ lies in $[0, 1]$. I^{cur} and I^{lib} need not have equal cardinality. Larger values of $BS(\cdot)$ mean more similarity/ closer match. The agent designer can decide what value of $BS(I, I', B, B')$ constitutes a 'match' between I and I'.

For a fixed pair of intention sets, $BS(I^{cur}, I^{lib}, B^{cur}, B^{lib}) \leqslant IS(I^{cur}, I^{lib})$. That is, $BS(\cdot)$ is a stronger measure of similarity than $IS(\cdot)$. This is because with $BS(\cdot)$, intention satisfaction levels must also be similar. The stronger measure is required to filter out b-plans that seem similar when judged only on the commonality of their intentions, but not on their satisfaction levels. And there may be several b-plans in the library which would be judged similar by $IS(\cdot)$, but they have been added to the library exactly because they are indeed different when their satisfaction levels are taken into account. The following example should make this clear. Suppose that the following two b-plans are in the library: $\{g1, g4\} : B_1 : \pi_1$ and $\{g1, g4\} : B_2 : \pi_2$, where $B_1 = \{(s_1, 0.95), (s_2, 0.05), (s_3, 0), (s_4, 0)\}$ and $B_2 = \{(s_1, 0),$

$(s_2, 0), (s_3, 0.05), (s_4, 0.95)\}$. And suppose $g1$ is most satisfied when the agent is in s_1, and $g4$ is most satisfied when the agent is in s_4. A policy to pursue $\{g1, g4\}$ when starting in B_1 would rather suggest actions to move towards s_4, while a policy to pursue $\{g1, g4\}$ when starting in B_2 would rather suggest actions to move towards s_1. The point is that although the two b-plans are identical with respect to the intention set, they have very different policies, due to their different belief states (and thus satisfaction levels).

The concept of KL divergence or cross-entropy (Kullback, 1968; Csiszár, 1975) could be used as a measure of similarity between two belief states. Although not as well supported by information theory as cross entropy, we use a simpler (more intuitive) definition of similarity and easier to implement. Also, the measure presented and used here is symmetrical and always defined, whereas cross-entropy may give different values for the similarity of B from B', and of B' from B, and may be undefined in some cases. We define the similarity function for belief states as

$$Sim_\beta(B, B') := 1 - \frac{1}{2}\sum_{s \in S} ||B(s) - B'(s)||.$$

4.4.3. Summary

To "execute policy π" (where π has horizon/depth h) means to perform up to h actions as recommended by π. 'Exhausting' a policy, that is, using all h actions per policy, may be inadvisable in many domains, because actions deeper in a policy will be less optimal. Suppose $1 \leqslant h^< \leqslant h$ is the agent-designer-defined number of actions the agent will execute per policy. No policy will be sought in the library, nor will a new policy be generated until the first $h^<$ action recommendations of the current policy have been executed. One may ask, If only $h^<$ actions are executed per policy, why not simply generate policies only to depth $h^<$ and exhaust those shorter policies? The answer is because then all actions executed will be worse estimates of what good actions are; the search depth beyond $h^<$ provides information which improves recommendations before and up to $h^<$.

One may be concerned that a policy becomes 'stale' or inapplicable while being executed, and that seeking or generating 'fresh' policies at every iteration keeps action selection relevant in a dynamic world. However, written policies (in a-plans) should preferably have the form of generated policies, and generated policies (in b-plans) can deal with all situations understood by the agent: It is assumed that each observation distinguishable by the agent, identifies a particular state of the world, as far as the agent's sensors allow. Hence, if a policy considers every observation at its choice nodes, the policy will have a recommended (for written policies) or optimal (for generated policies) action, no matter the state of the world. However, writing or generating policies with far horizons (e.g., $h > 7$) is impractical. With large h, an agent will take relatively long to generate a policy and thus lose its reactiveness. Reactiveness is especially important in highly dynamic environments.

Suppose that the agent currently has a belief state B^{cur} and an intention set I^{cur}. First, the agent will scan through all a-plans, selecting all those which 'match' I^{cur}. From this set, the agent will execute the policy π of the a-plan $I : \Phi : \pi$ whose attribute condition has the highest degree of belief at B^{cur}. If the set of a-plans matching I^{cur} is empty, the agent will scan through all b-plans, selecting all those which 'match' I^{cur}. From this set, the agent will execute the policy π of the b-plan $I : B : \pi$ whose belief state is 'most similar' to B^{cur}. If the set of b-plans matching I^{cur} is empty, or there is no b-plan with belief state similar to B^{cur}, then the agent will generate policy π^{cur}, execute it and store $I^{cur} : B^{cur} : \pi^{cur}$ in the plan library for possible reuse later. The high-level planning process is depicted by the diagram in Fig. 3.

Two thresholds are involved with determining when library plans are applicable and how plans are dealt with: the *intention-set threshold* (abbreviation: θ_i) and the *belief-state threshold* (abbreviation: θ_b). The former is involved in both a-plans and b-plans, and the latter is involved only in b-plans.

The *FindPolicy* procedure (Algo. 2) formally defines what policy the agent will execute whenever the agent seeks a policy, and the procedure defines when and how new plans are added to the plan library.

5. Evaluation of HPB agent performance

The vanilla POMDP planner is used as a baseline.[1] A set of experiments was run on the original HPB architecture for comparison with the extended version and with the vanilla POMDP planner.

First, the simulated domain is explained and specified, including the parameters for each of the three controllers (planners/architectures). Second, the results of the three controllers deployed in the domain are presented and analyzed for various parameter settings.

5.1. Simulation

We performed tests with a six-by-six (columns-by-rows) grid-world simulation, where the agent's task is to visit each of the four corners, and to collect twelve items randomly scattered. The architecture and domain are coded in Python; no agent programming language was used. Each of the three controllers was optimized to perform as well as possible (given the time constraints for this research). Before giving the details for each case, we present the details which are common to all.

States are quadruples $\langle x, y, d, i\rangle$, with $x, y \in \{1, \ldots, 6\}$ being the coordinates of the agent's position in the world, $d \in \{\texttt{North}, \texttt{East}, \texttt{West}, \texttt{South}\}$ the direction it is facing, and $i \in \{0, 1\}, i = 1$ if an item is present in the cell with

[1] Except for the mean-as-threshold method mentioned below.

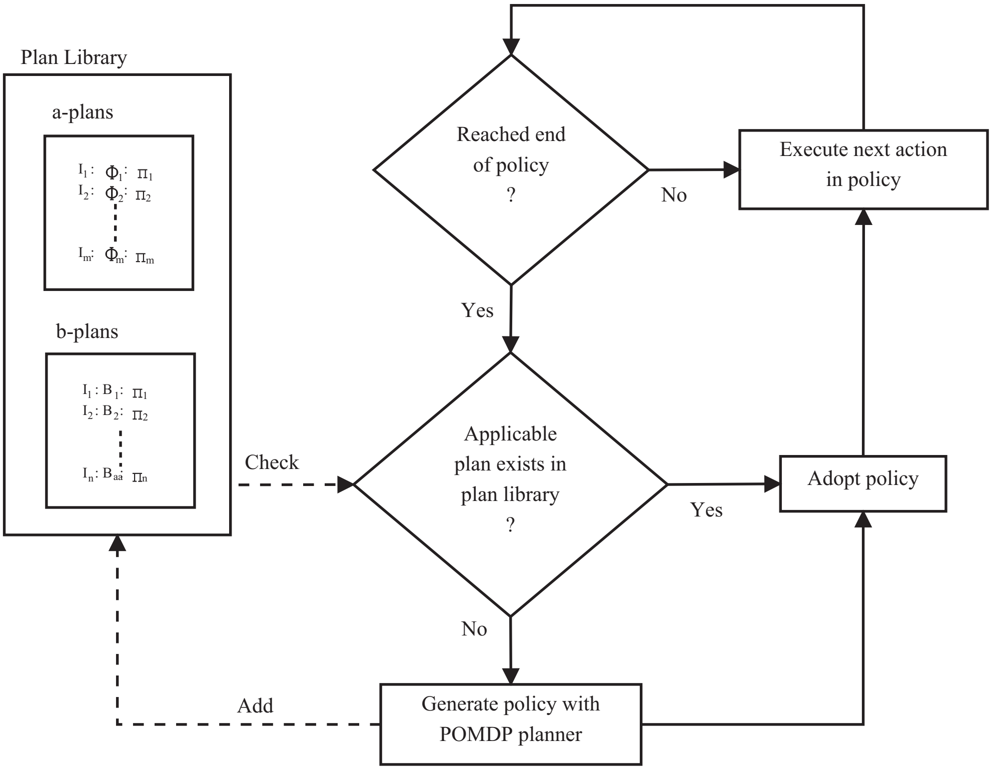


Fig. 3. A flow diagram of the planning process in the new version of the HPB agent architecture.

the agent, else $i = 0$. The agent can perform five actions $\{\texttt{left}, \texttt{right}, \texttt{forward}, \texttt{see}, \texttt{take}\}$, meaning, turn left, turn right, move one cell forward, see whether an item is present and take an item. The only observation possible when executing one of the physical actions is `obsNil`, the null observation, and `see` has possible observations from the set $\{0, 1\}$ for whether the agent sees the presence of an item (1) or not (0).

Next, we define the possible outcomes for each action: When the agent turns left or right, it can get stuck in the same direction, turn 90° or overshoots by 90°. When the agent moves forward, it moves one cell in the direction it is facing or it gets stuck and does not move. The agent can see an item or see nothing (no item in the cell), and taking is deterministic (if there is an item present, it will be collected with certainty, if the agent executes `take`). All actions except `take` are designed so that the correct outcome is achieved 95% of the time and incorrect outcomes are achieved 5% of the time.

The agent looks four steps into the future while planning ($h = 4$), unless stated otherwise. The discount factor is set to 0.9. Desire update rule (3) is used.

For each experiment, 30 trials were run and averages are reported. For each trial, the agent starts in a random location and performs between 100 and 400 actions. In order to make planning more efficient, all belief states were compressed using the *mean-as-threshold* method (Rens & Ferrein, 2013). The experiments were performed on a PC with an Intel® Core ™ i7-2630QM CPU @ 2.0 GHz and 8 GB RAM.

5.1.1. Vanilla POMDP planner

The reward function $R(a, s)$ returns

$$\alpha(1 - C_{dist}/10) + (1 - \alpha)(5(1 - T_{dist}/10 + E + 5)/11) - \mathit{FimInd}(x, y)/5.$$

Let's break it down:

- C_{dist} is the Manhattan distance between the cell represented by s and the closest corner cell.
- 10 is the maximum Manhattan distance between any two cells, for instance, moving from corner $(0, 0)$ to corner $(6, 6)$ requires ten forward actions.
- $1 - C_{dist}/10$ is the normalized inverse Manhattan distance between the cell represented by s and the closest corner cell.
- Similarly, $1 - T_{dist}/10$ is the normalized inverse Manhattan distance between the cell represented by s and the cell containing the closest item.
- α trades off the importance of visiting corners and collecting items.
- In $5(1 - T_{dist}/10 + E + 5)/11$, the first 5 was found to improve performance in general. The second 5, the E and the 11 are used in an attempt to normalize the term to the unit range so that the trade-off factor can be applied more reasonably.
- $\mathit{FimInd}(x, y)$ is the familiarity index of cell (x, y). It is incremented by 1 each time cell (x, y) is visited. Division by 5 was found to improve general performance. It has a similar motivation as the satisfaction function in the extended HPB architecture, but cruder.

- Term $-FimInd(x,y)/5$ thus steers the agent away from cells visited often. This is especially useful to stop the agent from getting stuck in corners.

The reader should notice the effort to optimize the agent's performance as much as possible while staying within the definition of regular POMDP theory: Many different parameters were tried and several hours spent on optimization. The reader's attention is also drawn to the ad hocness and complexity of the reward function.

5.1.2. Original HPB architecture

The goals are $G = \{(1,1),(1,6),(6,1),(6,6),\texttt{collect}\}$. The satisfaction function is designed to pursue corner visiting:

$Satf(g,s) = 1 - dist/10,$

where 10 is the maximum Manhattan distance between two cells in the world and *dist* is the Manhattan distance between the cells represented by g and s. The preference function is designed to make the agent collect items:

$Pref(a,s) = (1 - dist/10 + E)/6,$

where $E = 0$ unless a is the `take` action and there is (believed to be) an item in the cell corresponding to state s, in which case, $E = 5$. The threshold θ_f – amount of change in satisfaction levels, involved in determining the intention set at each iteration – is set to $0.05, MRY$ is set to 5. These two parameters are the same for the extended version. Desire levels are initially set to zero for all goals.

5.1.3. Extended HPB architecture

The goals are still $\{(1,1),(1,6),(6,1),(6,6),\texttt{collect}\}$, and the corner goals are marked mutually incompatible. That is,

- $Cpbl((1,1)) = \{(1,1),\texttt{collect}\}$,
- $Cpbl((1,6)) = \{(1,6),\texttt{collect}\}$,
- $Cpbl((6,1)) = \{(6,1),\texttt{collect}\}$,
- $Cpbl((6,6)) = \{(6,6),\texttt{collect}\}$,
- $Cpbl(\texttt{collect}) = \{\texttt{collect},(1,1),(1,6),(6,1),(6,6)\}$.

Experiments were performed with different weight-combinations ($W(g)$) assigned for each experiment. When $g \in \{(1,1),(1,6),(6,1),(6,6)\}$, we let

$\rho(a,g,s) = 1 - dist/10,$

where *dist* is the Manhattan distance between the cells represented by g and s – the action a is ignored when $g \in \{(1,1),(1,6),(6,1),(6,6)\}$. And we let

$\rho(a,\texttt{collect},s) = (1 - dist/10 + E + 5)/6,$

where *dist* is the Manhattan distance between the cell represented by s and the closest cell containing an item, and $E = 0$ unless a is the `take` action and there is (believed to be) an item in the cell corresponding to state s. The cost function κ always returns 0 for this domain. The satisfaction function $\sigma(g,s)$ is defined similarly to the reward function: it returns $(1 - dist/10 + E + 5)/11, E = 5$ if there is *no* item in s, else $E = -5$. In the reward function, the agent is rewarded for taking an item when there is one to take, whereas in the satisfaction function, the agent is more satisfied when there is no item in the cell, that is, when the environment is cleaner. Note that the values returned by ρ, κ and σ as defined here are in $[0,1]$. Otherwise, the specific design of ρ and σ are to maximize performance, that is, item collection and corner visiting.

The plan library only stored and reused b-plans; a-plans were not made use of. θ_i and θ_b were both set to 0.9.

5.2. Results

5.2.1. High-level comparison of the three controllers

Table 1 shows the results of the agent controlled by the vanilla POMDP planner. Recall that α is not formally a part of a POMDP model, but is used in order to compare vanilla POMDP planning to the other controllers in this grid-world simulation. α trades corner visiting off with item collection, "IC" indicates the average number of items collected, "(x,y)" indicated the average number of times corner (x,y) was visited and "Duration" is the average duration (in seconds) of the thirty trials for each of the α values. A single trial is 100 actions. Note that as the trade-off factor favours corner visiting, items collected decreases. However, corners visited does not change with α. There is thus difficulty in directing a POMDP planner to pursue particular tasks or goals without re-defining the reward function for each new task. That is, using a trade-off factor is insufficient. Moreover, focusing on individual corners would complicate the reward function even more than it already is. We shall refer to these results again in comparison to the two other controllers below.

Table 2 shows the results of the agent controlled by the original HPB architecture. Note that as the trade-off factor favours corner visiting, items collected decreases and corners visited increases. So this is an improvement on the vanilla POMDP controller. However, focusing on individual corners is still not possible in this architecture. The reason why the total number of corners visited does not increase (proportionally to α) for $\alpha > 0.5$ is because the agent is limited to 100 actions. Also note that trials take approximately half as long as with the vanilla POMDP controller. This must be due to a combination of the more modular definition of rewards in the HPB architecture and only one goal being pursued at a time. A more detailed analysis for the improvement in running time is beyond the scope of this work.

Table 3 shows the results of the agent controlled by the extended HPB architecture. The agent thus has a plan library with the ability to store policies and reuse them when appropriate. In this particular experiment, the agent performs two actions for every policy generated or reused, and the goals of visiting corners are marked as disjoint (see next subsection). Note that, in Table 3, the weight of importance for collecting items ($W(\texttt{collect})$) is

Table 1
Performance of vanilla POMDP planner; 100 actions per trial.

α	IC	(1, 1)	(1, 6)	(6, 1)	(6, 6)	Duration
0	9.5	1.0	1.0	1.0	1.1	240
0.25	8.4	1.2	1.1	1.5	1.2	281
0.5	4.9	1.7	1.5	1.6	1.5	272
0.75	2.1	1.6	1.7	1.8	1.8	284
1	0.1	1.8	1.8	2.0	1.9	278

Table 2
Performance of original HPB architecture; 100 actions per trial.

α	IC	(1, 1)	(1, 6)	(6, 1)	(6, 6)	Duration
0	9.7	0.2	0.3	0.2	0.1	162
0.25	7.0	1.2	1.3	1.1	1.4	150
0.5	2.6	2.2	2.3	1.9	2.4	125
0.75	1.3	3.0	2.5	2.4	2.5	121
1	0.3	2.3	2.3	2.3	2.5	130

Table 3
Performance of extended HPB architecture; 100 actions per trial.

$W(\texttt{collect})$	IC	(1, 1)	(1, 6)	(6, 1)	(6, 6)	Duration
1	9.7	0.4	0.4	0.4	0.5	41
0.75	9.5	0.9	0.8	1.0	0.6	72
0.5	6.6	1.2	1.2	0.9	1.2	122
0.25	2.6	2.6	2.2	2.2	2.2	58
0	0.2	3.4	3.2	3.4	3.3	88

analogous in meaning to $1 - \alpha$ with reference to the experiments reported in Tables 1 and 2. In the case of Table 3, the weights for corner visiting are set to $(1 - W(\texttt{collect}))/4$. Note that task duration improves significantly (on average) while overall performance remains the same as with the original HPB architecture (Table 2) or improves. The overall average task/trial durations are 271 (vanilla POMDP), 138 (original HPB) and 76 (extended HPB).

5.2.2. Goal compatibility

Next, we investigate whether making goals disjoint (when appropriate) actually improves performance. Tables 4 and 5 report on the same set of experiment designs, differing only on whether corner visiting is not disjoint (no joint goal pursuit restrictions), respectively, disjoint (as specified by the compatibility sets $Cpbl(\cdot)$ in Section 5.1.3). Weight assignments appear in columns with header "Wi", and the column just to the right – with header "Res." – gives the result for the goal in the corresponding row. Task/trial duration is essentially the same in both tables/-cases. We see from Table 5 that when it is specified that two or more corners may not be pursued simultaneously, the agent's behavior is more or less as desired, with one exception. In the case of the third weight assignment (W3), the agent hardly visits corners (1, 6) and (6, 1). This is likely because they are relatively far apart; in the case of W6, for instance, corners (1, 1) and (1, 6) are visited at a relatively high frequency. In the case where corner visiting is unrestricted, performance breaks down. For instance, in Table 4, only cases W1, W4 and W5 seem to have satisfactory performance. In summary, allowing for the specification of compatible/incompatible goals seems like a useful feature in an agent architecture with multiple goals. In the rest of the experiments, agents are specified to allow the pursuit of only a single corner at a time.

5.2.3. Effects of the plan library

In the previous experiments, a plan library was not used. In the rest of the experiments, a plan library is always used. However, the library contains no (written) a-plans, only (generated) b-plans are stored and reused. In the tables, "PL size" refers to the number of plans stored in the plan

Table 4
Performance of extended HPB architecture with unrestricted corner visiting, without a plan library; 100 actions per trial.

Goal	W1	Res.	W2	Res.	W3	Res.	W4	Res.	W5	Res.	W6	Res.	W7	Res.
(1, 1)	.0625	0.5	.1875	0.4	0	0.2	0	0.2	0	0.0	.5	0.8	.333	3.8
(1, 6)	.0625	0.3	.1875	0.3	.5	0.1	.3	1.6	.7	6.8	.5	0.5	.333	0.1
(6, 1)	.0625	0.5	.1875	0.2	.5	0.0	0	0.0	0	0.0	0	0.0	.333	0.0
(6, 6)	.0625	0.2	.1875	0.2	0	1.3	0	0.0	0	0.0	0	0.0	0	0.0
`collect`	.75	9.7	.25	6.1	0	2.8	.7	5.2	.3	1.8	0	0.5	0	1.1
Dur.		131		137		141		123		127		136		142

Table 5
Performance of extended HPB architecture with disjoint corner visiting, without a plan library; 100 actions per trial.

Goal	W1	Res.	W2	Res.	W3	Res.	W4	Res.	W5	Res.	W6	Res.	W7	Res.
(1, 1)	.0625	1.0	.1875	3.0	0	1.0	0	0.1	0	0.1	.5	3.4	.333	3.9
(1, 6)	.0625	0.8	.1875	2.4	.5	0.8	.3	2.0	.7	3.8	.5	3.8	.333	2.6
(6, 1)	.0625	0.4	.1875	2.6	.5	0.6	0	0.0	0	0.0	0	0.0	.333	2.3
(6, 6)	.0625	0.4	.1875	2.5	0	0.9	0	0.9	0	0.1	0	0.0	0	1.8
`collect`	.75	9.2	.25	2.1	0	0.6	.7	4.7	.3	1.3	0	0.1	0	0.0
Dur.		131		132		154		130		129		146		140

library at the end of a trial, and "# Reu." refers to the number of times any stored plan was retrieved (and reused) per trial. Recall that the look-ahead depth is by default $h = 4$ in our experiments. Table 6 reports on results where only the first action of a policy, whether freshly generated or retrieved from the library, is executed. After every single action, a new policy of depth 4 is generated or retrieved and only the first action executed. Table 7 reports on results where the only difference is that the first *two* actions are executed for every policy. We compare results in Table 5 (no library) with Table 7 (library; one action) with Table 8 (library; two actions).

The first thing to notice is that when policies are reused and single actions executed (Table 6) approximately 1/3 to 1/2 of cached plans are retrieved and reused, and a corresponding saving in task duration is accomplished over task duration when no plans are stored (Table 5). Also notice that there is no significant difference in task quality. Weight assignment 3 (W3) in Table 6 produces undesired performance. This seems to be due to how corner visiting is defined: a corner is visited if at the previous iteration, the agent was not in the particular corner. So if there is no reason for the agent to move away from the corner (to pursue other goals), then it will not move out of the corner to allow it to gain a reward for *re*visiting. This 'undesirable' behavior might actually be acceptable with a different definition of the goal, or should be solved with a more sophisticated reward function definition.

Table 6
Performance of extended HPB architecture; single action execution per policy; 100 actions per trial.

	W1	Res.	W2	Res.	W3	Res.	W4	Res.
(1, 1)	.0625	0.6	.1875	3.0	0	0.2	0	0.3
(1, 6)	.0625	0.5	.1875	2.3	1	1.0	0	0.2
(6, 1)	.0625	0.8	.1875	2.4	0	0.0	0	0.4
(6, 6)	.0625	0.7	.1875	2.6	0	0.0	0	0.2
collect	.75	8.7	.25	1.9	0	0.3	1	8.1
Dur.		116		109		16		72
PL size		87		78		12		50
# Reu.		13		22		88		50
	W5	Res.	W6	Res.	W7	Res.	W8	Res.
(1, 1)	0	1.5	0	1.5	0	0.2	0	0.2
(1, 6)	.5	1.1	.3	0.1	.3	2.6	.7	2.0
(6, 1)	.5	1.2	.7	3.4	0	0.0	0	0.0
(6, 6)	0	1.3	0	1.9	0	0.0	0	0.0
collect	0	0.1	0	0.3	.7	3.8	.3	0.9
Dur.		112		79		93		42
PL size		75		53		69		27
# Reu.		25		47		31		73

Table 7
Performance of extended HPB architecture; two actions executed per policy; 100 actions per trial.

	W1	Res.	W2	Res.	W3	Res.	W4	Res.
(1, 1)	.0625	0.9	.1875	2.6	0	0.2	0	0.4
(1, 6)	.0625	0.8	.1875	2.2	1	0.8	0	0.4
(6, 1)	.0625	1.0	.1875	2.2	0	0.0	0	0.4
(6, 6)	.0625	0.6	.1875	2.2	0	0.0	0	0.5
collect	.75	9.5	.25	2.6	0	0.3	1	9.7
Dur.		72		58		9		41
PL size		47		44		7		33
# Reu.		3		6		43		17
	W5	Res.	W6	Res.	W7	Res.	W8	Res.
(1, 1)	0	1.6	0	1.3	0	0.2	0	0.3
(1, 6)	.5	0.6	.3	0.2	.3	2.5	.7	2.9
(6, 1)	.5	0.6	.7	3.2	0	0.1	0	0.0
(6, 6)	0	1.1	0	1.6	0	0.1	0	0.0
collect	0	0.2	0	0.5	.7	4.8	.3	0.9
Dur.		55		47		65		37
PL size		40		35		39		22
# Reu.		10		15		11		28

When the agent uses one action per policy (Table 6), on average, trial duration is 80 s, library size is 56 and the number of reused plans is 44. When the agent uses two actions per policy (Table 7), on average, trial duration is 48 s, library size is 33 and the number of reused plans is 17. Hence, one can observe that increasing the number of actions used per policy decreases the number of policies which must be generated and thus decreases the task duration, while task quality seems unaffected (e.g., avg. items collected: 3.01 using one action per policy vs. 3.56 using two actions per policy). Although we did not investigate in detail the effect of using more/all h actions per policy, it is inadvisable to use more than, say, half the policy depth because actions deeper in a policy will be less optimal.

5.2.4. Further insights

To gain a better understanding of how look-ahead depth affects agent behavior, we performed a set of experiments with $h = 5$. (This is the only time when $h \neq 4$.) Two actions per policy were used, so these results (Table 8) should be compared with the results in Table 7. As expected, task quality improves at the cost of a significant increase in task duration.

To gain a deeper sense for the behavior of agents based on the extended HPD architecture, we performed a set of experiments where the agent is allowed to perform 200 actions. One action per policy was used, so these results (Table 9) should be compared with the results in Table 6. As would be expected, trial duration approximately doubles. Corners visited also doubles, but because there is a

Table 8
Performance of extended HPB architecture; $h = 5$; two actions executed per policy; 100 actions per trial.

	W1	Res.	W2	Res.	W7	Res.	W8	Res.
(1, 1)	.0625	0.9	.1875	2.8	0	0.2	0	0.1
(1, 6)	.0625	0.7	.1875	2.3	.3	2.7	.7	3.9
(6, 1)	.0625	0.8	.1875	2.4	0	0.1	0	0.0
(6, 6)	.0625	0.4	.1875	2.6	0	0.1	0	0.0
collect	.75	10.1	.25	3.3	.7	5.1	.3	1.4
Dur.		350		338		232		161
PL size		48		44		38		25
# Reu.		2		6		12		25

Table 9
Performance of extended HPB architecture; single action execution per policy; 200 actions per trial.

	W1	Res.	W2	Res.	W7	Res.	W8	Res.
(1,1)	.0625	1.7	.1875	5.8	0	0.2	0	0.1
(1,6)	.0625	1.5	.1875	5.5	.3	3.5	.7	5.0
(6,1)	.0625	1.3	.1875	5.5	0	0.0	0	0.0
(6,6)	.0625	1.3	.1875	5.9	0	0.0	0	0.0
collect	.75	9.2	.25	2.2	.7	4.6	.3	1.8
Dur.		191		156		178		114
PL size		161		130		109		68
# Reu.		39		70		91		132

limited number of items to collect (12 items), the number of items collected does not significantly increase. What is perhaps more interesting is that in the results of Table 6, the average percentage of plans reused is 79, whereas when double the amount of actions are performed (Table 9), the average percentage of plans reused is 71. Our intuition was that the percentage of reuse would increase monotonically with an increase in actions performed. However, it seems that the agent continued to find itself in unfamiliar situations. Percentage plan reuse does increase to 114 when 400 actions are allowed per trial; see Table 10. However, the variance in percentage per weight assignment case is large (Table 9). Although in W7 (Table 9), visiting corner $(1,6)$ has weight 0.3, whereas in W8 it has a weight of 0.7, the corner is visited approximately 5.2 times in both cases. This can be explained by noting that the 'pathological' behavior the agent displays when only one corner has non-zero weight is independent of the number of actions performed by the agent.

5.2.5. Summary

These experiments highlight some important features of the (latest) HPB architecture: It can be seen quite clearly that the agent can be directed to certain corners and to collect items with a dedication proportional to the weights chosen by the agent designer for the respective goals. Each of several goals can be pursued individually until satisfactorily achieved. Goals can be satisfied even while dealing with stochastic actions and perceptions. Policy reuse effectively reduces the time it takes to complete a task without affecting task quality. There are still some cases when the agent performs poorly, although the performance in these cases might be improved by more careful engineering.

Table 10
Performance of extended HPB architecture; single action execution per policy; 400 actions per trial.

	W1	Res.	W2	Res.	W7	Res.	W8	Res.
(1,1)	.0625	2.7	.1875	11.4	0	0.2	0	0.2
(1,6)	.0625	3.1	.1875	11.0	.3	5.3	.7	5.1
(6,1)	.0625	2.3	.1875	11.3	0	0.0	0	0.0
(6,6)	.0625	2.9	.1875	11.4	0	0.1	0	0.0
collect	.75	10.8	.25	3.5	.7	5.3	.3	1.7
Dur.		400		262		171		114
PL size		315		205		144		82
# Reu.		85		195		256		318

6. Related work

AgentSpeak$^+$ (Bauters et al., 2015) extends the BDI language AgentSpeak (Rao, 1996) with on-demand probabilistic planning in uncertain environments. AgentSpeak has a plan library of plans, each plan being of the form

$$e : b_1 \wedge \cdots \wedge b_m \leftarrow c_1; \cdots ; c_n$$

where e is a *triggering event*, $b_1, \ldots, b_m$ are belief literals and $c_1, \ldots, c_n$ are actions or goals. Goals may become (internal) triggering events. Events in the (external) environment may also be perceived as triggering events. As triggering events occur, they are placed in a set and periodically selected for processing. An event is 'processed' by selecting an appropriate plan from the plan library with a matching triggering event. A plan is appropriate if its *context* $b_1 \wedge \cdots \wedge b_m$ is a logical consequence of the agent's set of *base beliefs*. The goals and/or actions $c_1; \cdots ; c_n$ of the selected appropriate plan will be processed in sequence. If c_i is an action, it is executed; if it is a goal, it becomes an internal event which may trigger the selection and execution of further plans. An AgentSpeak agent maintains a set of intentions and each intention is a stack of plans. Please refer to (Rao, 1996) for details. When considering HPB plans, e is roughly analogous to I, $b_1 \wedge \cdots \wedge b_m$ is roughly analogous to Φ or B and $c_1; \cdots ; c_n$ is roughly analogous to π.

The contribution of AgentSpeak$^+$ is to allow a POMDP planner to suggest the optimal action at a point in a (written) plan where the agent designer feels that an *optimal* action is required at that point, or that there is insufficient information at the time of writing the plan to suggest a reasonable action. In other words, there might be points in a plan when actions are best chosen just before execution so that they can be determined appropriately for the agent's *current* context.

Bauters et al. (2015) make use of only the first action of any POMDP policy. Online POMDP planners do forward-search to a given depth h (number of future actions). The deeper the look-ahead depth, the more optimal the actions in the policy. It might actually be a waste of computational resources to discard the whole policy of depth h once it is available. An agent could use its whole policy-tree and only generate a new policy after it has finished using the current policy to execute h actions. However, the actions closer to the end of the policy tree will tend to be farther from optimal than those closer to the tree's root. In future work, we would like to find ways to balance out the myopic take-first-action approach and the over-optimistic take-all-actions approach.

AgentSpeak$^+$ does not have a mechanism for storing and reusing generated policies.

An advantage of AgentSpeak$^+$ is that their written plans can be more expressive than HPB plans: elements of their plans are written in a language based on a fragment of first-order logic, including n-ary predicates and variable terms. Nonetheless, even though an HPB a-plan is propositional in nature (not relational), a policy has a reasonably expressive tree structure with branching conditional on observations of context sentences. A desirable feature that AgentSpeak plans have that HPB plans lack is the ability to call plans from within plans.

Some slightly less related work will now be reviewed.

Walczak, Braubach, Pokahr, and Lamersdorf (2007) and Meneguzzi, Zorzo, Móra, and Luck (2007) have incorporated online plan generation into BDI systems, however the planners deal only with deterministic actions and observations.

Nair and Tambe (2005) use POMDP theory to coordinate teams of agents. However, their framework is very different to our architecture. They use POMDP theory to determine good role assignments of team members, not for generating policies online.

Lim, Dias, Aylett, and Paiva (2008) provide a rather sophisticated architecture for controlling the behavior of an emotional agent. Their agents reason with several classes of emotion and their agents are supposed to portray emotional behavior, not simply to solve problems, but to look believable to humans. Their architecture has a "continuous planner [...] that is capable of partial order planning and includes emotion-focused coping [...]" Their work has a different application to ours, however, we could take inspiration from them to improve the HPB architecture.

Pereira, Gonçalves, Dimuro, and Costa (2008) take a different approach to use POMDPs to improve BDI agents. By leveraging the relationship between POMDP and BDI models, as discussed by Simari and Parsons (2006), they devised an algorithm to extract BDI plans from optimal POMDP policies. The main difference to our work is that their policies are pre-generated and BDI-style rules are extracted for all contingencies. The advantage is that no (time-consuming) online plan/policy generation is necessary. The disadvantage of their approach is that all the BDI plans must be stores and every time the domain model changes, a new POMDP must be solved and the policy-to-BDI-plan algorithm must be run. It is not exactly clear from their paper (Pereira et al., 2008) how or when intentions are chosen. Although it is interesting to know the relationship between POMDPs and BDI models (Simari & Parsons,2006, 2011), we did not use any of these insights in developing our architecture. However, the fact that the HPB architecture does integrate the two frameworks, is probably due to the existence of the relationship.

Rens, Ferrein, and Van der Poel (2009) also introduced a hybrid POMDP-BDI architecture, but without a notion of desire levels or satisfaction levels. Although their basic approaches to combine the POMDP and BDI frameworks is the same as ours, there are at least three major differences: Firstly, they define their architecture in terms of the GOLOG agent language (Boutilier, Reiter, Soutchanski, & Thrun, 2000). Secondly, their approach uses a computationally intensive method for deciding whether to refocus; performing short policy look-aheads to ascertain the most valuable goal to pursue.[2] Our approach seems much more efficient. Thirdly, in their approach, the agent cannot pursue several goals concurrently.

Chen et al. (2013) incorporate probabilistic graphical models into the BDI framework for plan selection in stochastic environments. An agent maintains epistemic states (with random variables) to model the uncertainty about the stochastic environment, and corresponding belief sets of the epistemic state are defined. The possible states of the environment, according to sensory observations, and their relationships are modeled using probabilistic graphical models: The uncertainty propagation is carried out by Bayesian Networks and belief sets derived from the epistemic states trigger the selection of relevant plans from a plan library. For cases when more than one plan is applicable due to uncertainty in an agent's beliefs, they propose a utility-driven approach for plan selection, where utilities of actions are modeled in influence diagrams. Our architecture is different in that it does not have a library of pre-supplied plans; in our architecture, policies (plans) are generated online.

Although Nair and Tambe (2005) and Chen et al. (2013) call their approaches hybrid, our architecture can arguably more confidently be called hybrid because of its more intimate integration of POMDP and BDI concepts.

None of the approaches mentioned maintain desire levels for selecting intentions. The benefit of maintaining desire levels is that intentions are not selected only according what they offer with respect to their *current* expected reward, but also according to when last they were achieved.

7. Conclusion

Our work focuses on providing high-level decision-making capabilities for robots and agents who live in stochastic, noisy environments, where multiple goals and goal types must be pursued. We introduced a hybrid POMDP-BDI agent architecture, which may display emergent behavior, driven by the intensities of goal desires. In the past decade, several BDIAs have been augmented with capabilities to deal with uncertainty. The HPB architecture is novel in that it can pursue multiple goals concurrently. Goals must periodically be re-achieved, depending on the goals' desire levels, which change over time and in proportion to how close the goals are to being satisfied.

A major benefit of the HPB architecture is that every action recommended by a generated policy simultaneously maximizes the agent's reward with respect to pursuit of *all* the current intentions. As far as the authors are aware, no other agent architecture does this.

[2] Essentially, the goals in G are stacked in descending order of the value of $V^{*}_{HPB}(B, g, h^{-})$, where $h^{-} < h$ and B is the current belief state. The goal on top of the stack becomes the intention.

In previous work (Rens & Meyer, 2015), we argued that maintenance goals like avoiding moist areas (or collecting soil samples) should rather be viewed as a *preference* and modeled as a POMDP reward function. And specific tasks to complete (like collecting gas or keeping its battery charged) should be modeled as BDI desires. The idea is that while the agent is pursuing goals, it can concurrently perform rewarding actions not directly related to its goals. The architecture reported about in this paper does not make a clear distinction between overt and maintenance goals. In the new version of the architecture, that distinction can be simulated, however, now goals can be pursued in a much more fine-grained way via the choice of goal-weights ($W(g)$).

Another important feature brought into the new version is the ability to mark sets of goals as *disjoint* thereby forcing the agent to never pursue these goals concurrently, that is, disjoint goals will never be in I simultaneously. We showed that there are instances when specifying certain goals as being incompatible improves performance. It would be interesting to investigate whether and valuable if the agent could determine on its *own* which goals should not be pursued simultaneously.

Caching and reusing policies is an effective way of saving plan generation time. We saw in the experiments that our agent could perform actions up to 1.7 times faster (executing only the first action of a policy) with equivalent performance by reusing policies. And we also saw that policies are reused more, the more time the agent spends in the domain, resulting in more computation/time saving for longer tasks (in a local area).

Policies returned by *Plan* as defined in this paper are optimal (for finite horizon planning). A major benefit of a POMDP-based architecture is that the literature on POMDP planning optimization (Cai, Liao, & Carin, 2009; Li, Cheung, & Liu, 2005; Murphy, 2000; Paquet, Tobin, & Chaib-draa, 2005; Ross et al., 2008; Roy, Gordon, & Thrun, 2005; Shani, Brafman, & Shimony, 2007; Shani, Pineau, & Kaplow, 2013) (for instance) can be drawn upon to improve the speed with which policies can be generated.

The expressivity of the language we use for describing goals and for writing conditions in a-plans is relatively low. AgentSpeak, for instance, has a richer language. The language's expressivity is mostly independent of the architecture. We thus chose to use a simple language to better focus on the components we want to discuss.

In particular, HPB agents with (pre-written) a-plans included in the plan library must still be assessed. There is also scope for improving the focussing procedure. And analyzing under what conditions the two forms of desire update rule produce better performance must be investigated. We could take some advice from Antos and Pfeffer (2011). They provide a systematic methodology to incorporate emotion into a decision-theoretic framework, and also provide "a principled, domain-independent methodology for generating heuristics in novel situations".

There may be better methods for learning than policy reuse. Policy reuse has its place when reasoning time or power is limited, but given the time and power, more sophisticated techniques could perhaps generate and store shorter, more effective plans. For instance, when an agent encounters a landmark with relatively high certainty, the landmark's location can be stored. The agent could then augment its sensor readings with the stored location data to reach the landmark more easily in future. Some objects in the environment might not be stable, and their location data should 'degrade' over time in proportion to the environment's dynamism.

Singh, Sardina, Padgham, and James (2011) provide a method for learning which (pre-written) plans in a BDI system should be executed in which contexts (given a selection of context-applicable plans). Their approach can also relearn context-plan matches as conditions change in dynamic environments. Future versions of the HPB architecture could benefit from ideas in their work.

Prediction is an inherent part of POMDP planning, but we would like our agents to predict much farther into the future, and recognize critical events which it should deal with or avoid. POMDP policies and pre-written plans are more for local 'tactical' control. We need to bring in techniques for the agent to think globally or 'strategically'.

The set of intentions might change while executing a policy. If the current set of intentions changes a lot, the current policy might become inapplicable. This is a typical BDI *reconsideration* issue. However, an HPB agent will usually only perform very few actions before seeking a new plan. Just as in the case with humans, our agent should normally not get in trouble by assuming that things have not changed significantly in the last few steps. If the environment is so dynamic that relatively short plans can become inappropriate before completion of the plans, then the agent should have some more low-level, reactive systems to deal with the changes. In highly dynamical environments, the HPB 'agent' is better suited to being the high-level reasoning module of a larger system.

The design of the HPB agent architecture is a medium-to-long-term programme. We would like to keep improving its capabilities to deal with unforeseen, complex events in a changing, noisy environment. The next step is to rigorously test the architecture using an HPB agent in more complex simulated worlds.

References

Antos, D., & Pfeffer, A. (2011). Using emotions to enhance decision-making. In T. Walsh (Ed.), *Proceedings of the twenty-second Intl. Joint Conf. on Artif. Intell. (IJCAI-11)* (pp. 24–30). Menlo Park, CA: AAAI Press.

Bauters, K., McAreavey, K., Hong, J., Chen, Y., Liu, W., Godo, L., & Sierra, C. (2015). Probabilistic planning in agentspeak using the pomdp framework. In I. Hatzilygeroudis, V. Palade, & J. Prentzas (Eds.), *Combinations of intelligent methods and applications: Proceedings of the fourth intl. Workshop, CIMA 2014. Smart innovation, systems and technologies* (Vol. 46). Springer.

Boutilier, C., Reiter, R., Soutchanski, M., & Thrun, S. (2000). Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings of the seventeenth natl. conf. on artif. intell. (AAAI-00) and of the twelfth conf. on innovative applications of artif. intell. (IAAI-00)* (pp. 355–362). Menlo Park, CA: AAAI Press.

Bratman, M. (1987). Intention, plans, and practical reason. Massachusetts/England: Harvard University Press.

Cai, C., Liao, X., & Carin, L. (2009). Learning to explore and exploit in POMDPs. In *NIPS* (pp. 198–206).

Chen, Y., Hong, J., Liu, W., Godo, L., Sierra, C., & Loughlin, M. (2013). Incorporating PGMs into a BDI architecture. In G. Boella, E. Elkind, B. Savarimuthu, F. Dignum, & M. Purvis (Eds.), *PRIMA 2013: Principles and practice of multi-agent systems. Lecture notes in computer science* (Vol. 8291, pp. 54–69). Berlin/Heidelberg: Springer.

Csiszár, I. (1975). I-divergence geometry of probability distributions and minimization problems. *Annals of Probability, 3*, 146–158.

Kaelbling, L., Littman, M., & Cassandra, A. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence, 101* (1–2), 99–134.

Kinny, D., & Georgeff, M. (1991). Commitment and effectiveness of situated agents. In *Proceedings of the 12th Intl. Jconf. on Artif. Intell. (IJCAI-91)* (pp. 82–88).

Kinny, D., & Georgeff, M. (1992). Experiments in optimal sensing for situated agents. In *Proceedings of the second Pacific Rim Intl. Conf. on Artif. Intell. (PRICAI-92)*. .

Koenig, S. (2001). Agent-centered search. *Artificial intelligence Magazine, 22*, 109–131.

Kullback, S. (1968) (2nd ed.). *Information theory and statistics* (Vol. 1). New York: Dover.

Li, X., Cheung, W., & Liu, J. (2005). Towards solving large-scale POMDP problems via spatio-temporal belief state clustering. In *Proceedings of IJCAI-05 workshop on Reasoning with Uncertainty in Robotics (RUR-05)*. .

Lim, M., Dias, J., Aylett, R., & Paiva, A. (2008). Improving adaptiveness in autonomous characters. In J. Lester, H. Prendinger, & M. Ishizuka (Eds.), *Intelligent virtual agents. Lecture notes in computer science* (Vol. 5208, pp. 348–355). Berlin/Heidelberg: Springer.

Lovejoy, W. (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research, 28*, 47–66.

Meneguzzi, F., Zorzo, A., Móra, M., & Luck, M. (2007). Incorporating planning into BDI systems. *Scalable Computing: Practice and Experience, 8*(1), 15–28.

Monahan, G. (1982). A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science, 28*(1), 1–16.

Murphy, R. (2000). Introduction to AI Robotics. Massachusetts/England: MIT Press.

Nair, R., & Tambe, M. (2005). Hybrid bdi-pomdp framework for multiagent teaming. *Journal of Artificial Intelligence Research (JAIR), 23*, 367–420.

Paquet, S., Tobin, L., & Chaib-draa, B. (2005). Real-time decision making for large POMDPs. In *Advances in artif. intell.: Proceedings of the eighteenth conf. of the canadian society for computational studies of intelligence. Lecture notes in computer science* (Vol. 3501, pp. 450–455). Springer-Verlag.

Pereira, D., Gonçalves, L., Dimuro, G., & Costa, A. (2008). Constructing BDI plans from optimal POMDP policies, with an application to agentspeak programming. In G. Henning, M. G. and S. Goneet, (Eds.), *XXXIV Conferência Latinoamericano de Informática, Santa Fe. Anales CLEI 2008* (pp. 240–249).

Pollack, M., & Ringuette, M. (1990). Introducing the Tileworld: Experimentally evaluating agent architectures. In *Proceedings of the eighth conf. on artif. intell* (pp. 183–189). AAAI Press.

Rao, A. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In *Proceedings of the 7th European workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-96)* (pp. 42–55). Berlin/Heidelberg: Springer-Verlaag.

Rao, A., & Georgeff, M. (1995). BDI agents: From theory to practice. In *Proceedings of the ICMAS-95* (pp. 312–319). AAAI Press.

Rens, G., & Ferrein, A. (2013). Belief-node condensation for online POMDP algorithms. In *Proceedings of IEEE AFRICON 2013* (pp. 1270–1274). Red Hook, NY 12571 USA: Institute of Electrical and Electronics Engineers, Inc..

Rens, G., Ferrein, A., & Van der Poel, E. (2009). A BDI agent architecture for a POMDP planner. In G. Lakemeyer, L. Morgenstern, & M.-A. Williams (Eds.), *Proceedings of the ninth intl. symposium on logical formalizations of commonsense reasoning (Commonsense 2009)* (pp. 109–114). University of Technology, Sydney: UTSe Press.

Rens, G., & Meyer, T. (2015). Hybrid POMDP-BDI: An agent architecture with online stochastic planning and desires with changing intensity levels. In J. Van den Herik, B. Duval, J. Filipe, & S. Loiseau (Eds.), *Proceedings of the seventh Intl. Conf. on Agents and Artif. Intell. (ICAART), revised selected papers, LNAI* (pp. 79–99). Springer Verlaag.

Ross, S., Pineau, J., Paquet, S., & Chaib-draa, B. (2008). Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research (JAIR), 32*, 663–704.

Roy, N., Gordon, G., & Thrun, S. (2005). Finding approximate POMDP solutions through belief compressions. *Journal of Artificial Intelligence Research (JAIR), 23*, 1–40.

Schut, M., & Wooldridge, M. (2000). Intention reconsideration in complex environments. In *Proceedings of the fourth intl. conf. on autonomous agents (AGENTS-00)* (pp. 209–216). New York, NY, USA: ACM.

Schut, M., & Wooldridge, M. (2001). The control of reasoning in resource-bounded agents. *The Knowledge Engineering Review, 16*(3), 215–240.

Schut, M., Wooldridge, M., & Parsons, S. (2004). The theory and practice of intention reconsideration. *Experimental and Theoretical Artificial intelligence, 16*(4), 261–293.

Shani, G., Brafman, R., & Shimony, S. (2007). Forward search value iteration for POMDPs. In R. L. de Mantaras (Ed.), *Proceedings of the twentieth intl. Joint Conf. on Artif. Intell. (IJCAI-07)* (pp. 2619–2624). Menlo Park, CA: AAAI Press.

Shani, G., Pineau, J., & Kaplow, R. (2013). A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems, 27*(1), 1–51.

Simari, G., & Parsons, S. (2006). On the relationship between MDPs and the BDI architecture. In *Proceedings of the fifth intl. joint conf. on autonomous agents and multiagent systems, AAMAS '06* (pp. 1041–1048). New York, NY, USA: ACM.

Simari, G., & Parsons, S. (2011). *Markov decision processes and the belief-desire-intention model. Springer briefs in computer science*. New York, Dordrecht, Heidelberg, London: Springer.

Singh, D., Sardina, S., Padgham, L., & James, G. (2011). Integrating learning into a BDI agent for environments with changing dynamics. In T. Walsh (Ed.), *Proceedings of the twenty-second Intl. Joint Conf. on Artif. Intell. (IJCAI-11)* (pp. 2525–2530). Menlo Park, CA: AAAI Press.

Walczak, A., Braubach, L., Pokahr, A., & Lamersdorf, W. (2007). Augmenting BDI agents with deliberative planning techniques. In R. Bordini, M. Dastani, J. Dix, & A. Seghrouchni (Eds.), *Proceedings of the fourth intl. workshop of Programming Multi-Agent Systems (ProMAS-06)* (pp. 113–127). Heidelberg/Berlin: Springer-Verlag.

Wooldridge, M. (1999). Intelligent agents. In G. Weiss (Ed.), *Multiagent systems: A modern approach to distributed artif. intell*. Massachusetts/England: MIT Press.

Wooldridge, M. (2000). *Reasoning about rational agents*. Massachusetts/England: MIT Press.

Wooldridge, M. (2002). *An introduction to multiagent systems*. Chichester, England: John Wiley & Sons.