Branching Time Active Inference

empirical study and complexity class analysis

Théophile Champion TMAC3@KENT.AC.UK

University of Kent, School of Computing

Canterbury CT2 7NZ, United Kingdom

Howard Bowman H.BOWMAN@KENT.AC.UK

University of Birmingham, School of Psychology,

Birmingham B15 2TT, United Kingdom

University of Kent, School of Computing

Canterbury CT2 7NZ, United Kingdom

Marek Grześ M.GRZES@KENT.AC.UK

University of Kent, School of Computing

Canterbury CT2 7NZ, United Kingdom

Editor:

Abstract

Active inference is a state-of-the-art framework for modelling the brain that explains a wide range of mechanisms such as habit formation, dopaminergic discharge and curiosity. However, recent implementations suffer from an exponential (space and time) complexity class when computing the prior over all the possible policies up to the time horizon. Fountas et al (2020) used Monte Carlo tree search to address this problem, leading to very good results in two different tasks. Additionally, Champion et al (2021a) proposed a tree search approach based on (temporal) structure learning. This was enabled by the development of a variational message passing approach to active inference (Champion et al, 2021b), which enables compositional construction of Bayesian networks for active inference. However, this message passing tree search approach, which we call branching-time active inference (BTAI), has never been tested empirically. In this paper, we present an experimental study of the approach (Champion et al, 2021a) in the context of a maze solving agent. In this context, we show that both improved prior preferences and deeper search help mitigate the vulnerability to local minima. Then, we compare BTAI to standard active inference (AcI) on a graph navigation task. We show that for small graphs, both BTAI and AcI successfully solve the task. For larger graphs, AcI exhibits an exponential (space) complexity class, making the approach

intractable. However, BTAI explores the space of policies more efficiently, successfully scaling to larger graphs. Then, BTAI was compared to the POMCP algorithm (Silver and Veness, 2010) on the frozen lake environment. The experiments suggest that BTAI and the POMCP algorithm accumulate a similar amount of reward. Also, we describe when BTAI receives more rewards than the POMCP agent, and when the opposite is true. Finally, we compared BTAI to the approach of Fountas et al (2020) on the dSprites dataset, and we discussed the pros and cons of each approach.

Keywords: Active Inference, Variational Message Passing, Tree Search, Planning, Free Energy Principle

1. Introduction

Active inference extends the free energy principle (Friston, 2010; Pitti et al, 2020) to generative models with actions (Friston et al, 2016a; Da Costa et al, 2020; Champion et al, 2021b) and can be regarded as a form of planning as inference (Botvinick and Toussaint, 2012). This framework has successfully explained a wide range of brain phenomena, such as habit formation (Friston et al, 2016a), Bayesian surprise (Itti and Baldi, 2009), curiosity (Schwartenbeck et al, 2018), and dopaminergic discharge (FitzGerald et al, 2015). It has also been applied to a variety of tasks such as navigation in the Animal AI environment (Fountas et al, 2020), robotic control (Pezzato et al, 2020; Sancaktar et al, 2020; Wirkuttis and Tani, 2021), multi-vehicle control (Butz et al, 2019), the mountain car problem (Catal, Ozan and Verbelen, Tim and Nauta, Johannes and De Boom, Cedric and Dhoedt, Bart, 2020), the game DOOM (Cullen et al, 2018) and the cart-pole problem (Millidge, 2019).

Active inference builds on a subfield of Bayesian statistics called variational inference (Fox and Roberts, 2012), in which the true posterior is approximated with a variational distribution. This method provides a way to balance the computational cost and accuracy of the posterior distribution. Indeed, the variational approach is only tractable because some statistical dependencies are ignored during the inference process, i.e., the variational distribution is generally assumed to fully factorise, leading to the well known mean-field approximation:

$$Q(X) = \prod_i Q(X_i)$$

where X is the set of all hidden variables of the model and X_i represents the i-th hidden variable. Winn and Bishop (2005) presented a message-based implementation of variational inference, naturally called variational message passing. And more recently, Champion et al (2021b) rigorously framed active inference as a variational message passing procedure. By combining the Forney factor graph formalism (Forney, 2001) with the method of Winn and Bishop (2005), it becomes possible to create modular implementations of active inference (van de Laar and de Vries, 2019; Cox et al, 2019) that allows the users to define their own generative models without the burden of deriving the update equations. This paper uses a new software package called Homing Pigeon

that implements such a modular approach and the relevant code has been made publicly available on GitHub: `https://github.com/ChampiB/Homing-Pigeon`.

Arguably, the major bottleneck for scaling up the active inference framework was the exponential growth of the number of policies. In the reinforcement learning literature, this explosion is frequently handled using Monte Carlo tree search (MCTS) (Silver et al, 2016; Browne et al, 2012; Schrittwieser et al, 2019). MCTS is based on the upper confidence bound for trees (UCT), which originally comes from the multi-armed bandit problem, and trades-off exploration and exploitation during the tree search. In the reinforcement learning litterature, the selection of the node to expand is carried out using the UCT criterion[1], which is defined as:

$$UCT(s,a) = q(s,a) + C_{\text{explore}} \frac{P(s,a)}{1 + N(s,a)}, \tag{1}$$

where $q(s,a)$ is the value of taking action a in state s (i.e. q here is not the variational posterior), C_{explore} is the exploration constant that modulates the amount of exploration, $N(s,a)$ is the visit count, and $P(s,a)$ is the prior probability of selecting action a in state s. This approach has been applied to active inference in several papers (Fountas et al, 2020; Maisto et al, 2021). Fountas et al (2020) chose to modify the original criterion used during the node selection step that returns the node to be expanded. From equation (9) of (Fountas et al, 2020), one can see that the UCT formula has been replaced by:

$$U(s,a) = -\tilde{G}(s,a) + C_{\text{explore}} \frac{Q(a|s)}{1 + N(s,a)} \tag{2}$$

where $U(s,a)$ indicates the utility of selecting action a in state s; $N(s,a)$ is the number of times that action a was explored in state s; C_{explore} is an exploration constant equivalent to C_p in the UCT criterion; $Q(a|s)$ is a neural network modelling the posterior distribution over actions, which is trained by minimizing the variational free energy and $\tilde{G}(s,a)$ is the best estimation of the expected free energy (EFE) computed from the following equation:

$$\begin{aligned} G(\pi,\tau) = &- \mathbb{E}_{Q(\theta|\pi)Q(s_\tau|\theta,\pi)Q(o_\tau|s_\tau,\theta,\pi)}\Big[\ln P(o_\tau|\pi)\Big] \\ &+ \mathbb{E}_{Q(\theta|\pi)}\Big[\mathbb{E}_{Q(o_\tau|\theta,\pi)} H(s_\tau|o_\tau,\pi) - H(s_\tau|\pi)\Big] \\ &+ \mathbb{E}_{Q(\theta|\pi)Q(s_\tau|\theta,\pi)} H(o_\tau|s_\tau,\theta,\pi) - \mathbb{E}_{Q(s_\tau|\pi)} H(o_\tau|s_\tau,\pi), \end{aligned}$$

1. This version of UCT comes from Silver et al (2016)

using sampling of 3 (out of 4) neural networks[2] used by the system. Note that $Q(a|s)$ in equation (2) specializes $P(s,a)$ in equation (1), by providing the probability of selecting action a in state s. One can see that $U(s,a)$ in equation (2) has been obtained from UCT in equation (1), by replacing the average reward by the negative EFE.

More recently, Champion et al (2021a) proposed an online method that frames planning as a form of (temporal) structure learning guided by the expected free energy. This method, called branching-time active inference (BTAI), generalises active inference (Friston et al, 2016a; Champion et al, 2021b; Da Costa et al, 2020) and relates to another recently introduced framework for inference and decision making, called sophisticated inference (Friston et al, 2021). Importantly, the generative model of BTAI enables the agent to trade off risk and ambiguity, instead of only seeking for certainty as was the case in (Champion et al, 2021b). In this paper, we provide an empirical study of BTAI, enabling us to explicitly demonstrate that BTAI provides a more scalable realization of planning as inference than active inference.

Section 2 reviews the BTAI theory, with full details presented in (Champion et al, 2021a). Then, Section 3 compares BTAI to standard active inference in the context of a graph navigation task both empirically and theoretically. We show that active inference is able to solve small graphs but suffers from an exponential (space and time) complexity class that makes the approach intractable for bigger graphs. In contrast, BTAI is able to search the space of policies efficiently and scale to bigger graphs. Next, Section 4.2 presents the challenge of local minima in the context of a maze solving task, and shows how better prior preferences and deeper tree search help to overcome this challenge. Lastly, Section 4.3 compares two cost functions, $g^{classic}$ and g^{pcost}, in two new mazes. In Section 5, BTAI was compared to the POMCP algorithm (Silver and Veness, 2010) on the frozen lake environment; and the experiments suggest that BTAI and the POMCP algorithm accumulate a similar amount of reward. Also, we describe when BTAI receives more rewards than the POMCP agent, and when the opposite is true. In Section 6, BTAI was compared to the approach of Fountas et al (2020) on the dSprites dataset, and we discussed the pros and cons of each approach. Finally, Section 7 concludes this paper, and provides ideas for future research.

2. Branching Time Active Inference (BTAI)

In this section, we provide a short review of BTAI, and the reader is referred to (Champion et al, 2021a) for details. BTAI frames planning as a form of (temporal) structure learning guided by the expected free energy. This form of structure learning should not be confused with representational or parametric structure learning that is currently developped in the literature (Smith et al, 2020; Friston et al, 2016b; Friston et al, 2018). The idea is to define a generative model that can be expanded dynamically as shown in Figure 1.

2. Fountas et al (2020) used neural networks to model the likelihood mapping $P(o_\tau|s_\tau)$, the transition mapping $P(s_{\tau+1}|s_\tau,a_\tau)$, the posterior over states $Q(s_\tau)$, and the posterior over actions $Q(a_\tau|s_\tau)$

The past and present is modelled using a partially observable Markov decision process (POMDP) in which each observation (O_τ) only depends on the state at time τ, and this state (S_τ) only depends on the previous state ($S_{\tau-1}$) and previous action ($U_{\tau-1}$). In addition to the POMDP which models the past and present, the future is modelled using a tree-like generative model whose branches are dynamically expanded. Each branch of the tree corresponds to a trajectory of states reached under a specific policy. The branches are expanded following a logic similar to the Monte Carlo tree search algorithm (see below), and the state estimation is performed using variational message passing (Winn and Bishop, 2005; Champion et al, 2021b; Friston et al, 2017).

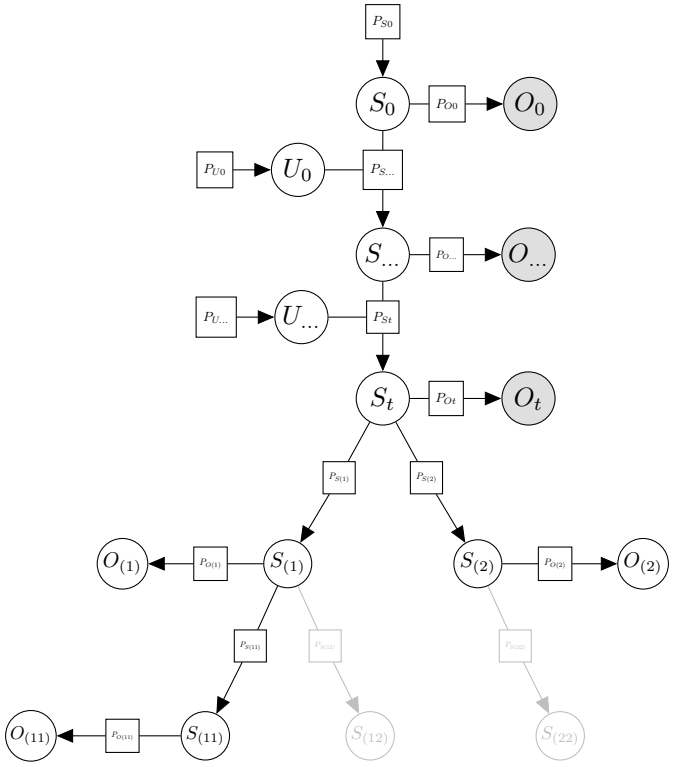


Figure 1: This figure illustrates the expandable generative model allowing planning under active inference. The current time point (the present) is denoted by t. All times before t are the past, and after t are the future. States in the future are indexed by multi-index (action sequences), with each digit indicating an action, e.g. $S_{(11)}$. The future is a tree-like generative model whose branches correspond to the policies considered by the agent. The branches can be dynamically expanded during planning and the nodes in light gray represent possible expansions of the current generative model.

At the start of a trial, the model contains only the initial hidden state S_0 and the initial observation O_0. Then, the agent starts expanding the generative model using an approach inspired by Monte Carlo tree search (Browne et al, 2012), where the selection of a node is based on expected free energy. More precisely, the node selection is performed recursively from the root until reaching a leaf node. At each level in the recursion the selected node maximises the UCT criterion:

$$UCT_J = \underbrace{-\bar{g}_J}_{\text{exploitation}} + \underbrace{C_p\sqrt{\frac{\ln n}{n_J}}}_{\text{exploration}},$$

where J is a multi-index representing a sequence of actions, S_J is the hidden state reached after performing the actions sequence described by the multi-index J, n is the number of times the parent of S_J has been visited, n_J is the number of times the child (S_J) was selected, and $\bar{g}_J$ is the average cost received after selecting S_J. In what follows, we denote by $J :: U$ the the multi-index obtained by adding the action U at the end of the sequence of actions described by the multi-index J. Once a leaf node (S_J) is selected for expansion, all its children states (i.e., all $S_{J::U}$) are added to the generative model. The future observations (i.e., $O_{J::U}$) associated to those hidden states (i.e., all $S_{J::U}$) are also added to the generative model. Next, the evaluation step estimates the cost of each state-observation pair $(S_{J::U}, O_{J::U})$. In this paper, we consider two kinds of cost. First, the standard expected free energy that trades off risk (over observations) and ambiguity:

$$g_J^{classic} \triangleq D_{\mathrm{KL}}[Q(O_J)||V(O_J)] \; + \; \mathbb{E}_{Q(S_J)}[\mathrm{H}[P(O_J|S_J)]],$$

where $J = I :: U$ for an arbitrary action U, and $V(O_J)$ is a distribution encoding the prior preferences over observations of the agent, which is generally parameterized by a vector $\boldsymbol{C}$ or learnt using a Dirichlet prior (Sajid et al, 2021). Second, we also experiment with the following quantity:

$$g_J^{pcost} \triangleq D_{\mathrm{KL}}\left[\, Q(S_J)||\, V(S_J)\right] + D_{\mathrm{KL}}\left[\, Q(O_J)||\, V(O_J)\right],$$

where $V(S_J)$ is a distribution encoding the prior preferences of the agent over the environment's states. Note that g_J^{pcost} depends on both the risk over observations and the risk over states. The reader is referred to Appendix B for a derivation of g_J^{pcost} from the Free Energy of the Expected Future (FEEF) introduced by Millidge et al (2021). Lastly, the cost of the best action (i.e., the action that produces the smallest cost) is propagated towards the root and used to update the aggregated cost of the ancestors of S_J.

Finally, during the planning procedure, the agent needs to perform inference of the future hidden states and observations. This is performed using variational message passing (VMP) on the set of newly expanded nodes, i.e. $\left\{S_{I::U}, O_{I::U} \mid U \in \{1, ..., |U|\}\right\}$, until convergence to a minimum in the free energy landscape. We refer the interested reader to (Champion et al, 2021b) for additional information about the derivation of the update equations. Also, since this paper only considers inference and not learning (i.e. the model does not have Dirichlet priors over the tensors defining the world's contingencies), the generative model is different from the one presented in the theoretical paper (Champion et al, 2021a). Therefore, we provide a mathematical description of the

generative model, the variational distribution and the belief updates in Appendix A. We summarise our method using the pseudo-code in Algorithm 1.

Algorithm 1: Branching Time Active Inference

```
while end of trial not reached do
    sample an observation (O_t) from the environment;
    perform inference using VMP and the newly acquired observation (O_t);
    while maximum planning iteration not reached do
        select a node to be expanded using the UCT criterion;
        perform the expansion of the generative model from the selected node;
        perform inference on the newly expanded nodes using VMP;
        evaluate the cost of the newly expanded nodes using g_J^classic or g_J^pcost;
        back-propagate the cost of the nodes through the tree;
    end
    select an action to be performed;
    execute the action in the environment;
end
```

3. BTAI vs active inference

In this section, we benchmark BTAI against standard active inference as implemented in Statistical Parametric Mapping (SPM), c.f. Friston (2007) for additional details about SPM. First, we do this in terms of complexity class and then empirically through experiments of increasing difficulty.

3.1 BTAI vs active inference: Space and Time complexity

In this section, we compare our model to the standard model of active inference (Friston et al, 2016a; Da Costa et al, 2020). In the standard formulation, the implementation needs to store the parameter of the posterior over states $\boldsymbol{s}_\tau^\pi$ for each policy and each time step. Therefore, assuming $|U|$ possible actions, T time steps, $|\pi| = |U|^T$ policies, and $|S|$ possible hidden state values, the space complexity class for storing the parameters of the posterior over hidden states is $\mathcal{O}(|\pi| \times T \times |S|)$. This corresponds to the number of parameters that needs to be stored, and it is a problem because $|\pi|$ grows exponentially with the number of time steps. Additionally, performing inference on an exponential number of parameters will lead to an exponential time complexity class.

BTAI solves this problem by allowing only K expansions of the tree. In BTAI, we need to store $|S|$ parameters for each time step in the past and present, and for each expansion, we only need to compute and store the parameters of the posterior over the hidden states corresponding to this expansion. Therefore, the time and space

complexity class is $\mathcal{O}([K+t] \times |S|)$, where t is the current time point. This is linear in the number of expansions. Now, the question is how many expansions are required to solve the task? Even if the task requires the tree to be fully expanded, then the complexity class of BTAI would be $\mathcal{O}\big([|U|^{T-t}+t] \times |S|\big)$. Figure 2 illustrates the difference between AcI and BTAI in terms of the space complexity class, when BTAI performs a full expansion of the tree.

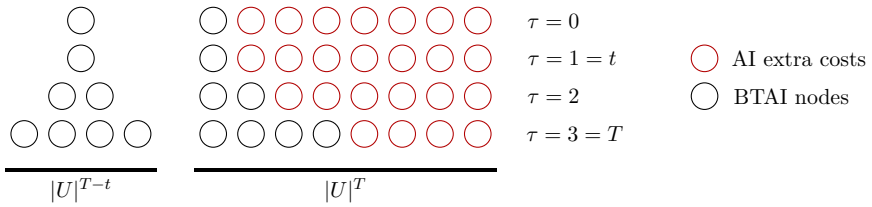


Figure 2: This figure illustrates the difference between AcI and BTAI in terms of space complexity class. The time goes from top to bottom, we assume two actions at each time step, t denotes the current time point, and each circle represents the storage of the $|S|$ parameters required to store a categorical distribution of a hidden state. Black nodes represent the nodes that must be stored in BTAI (under a full expansion of the tree), while the red nodes represent AcI's extra costs of storage. This extra cost comes from the fact that in AcI, one needs to store posterior beliefs for each time step and for each policy, while in BTAI, the tree allows us to compress the representation.

Additionally to the gain afforded by the structure of the tree, most practical applications can be solved by expanding only a small number of nodes (Silver et al, 2016; Schrittwieser et al, 2019), which means that MCTS and BTAI approaches will be even more optimised than in Figure 2 because most branches will not be expanded.

One could argue that there is a trade off in the nature and extent of the information inferred by classic active inference and branching-time active inference. Specifically, classic active inference exhaustively represents and updates all possible policies, while branching-time active inference will typically only represent a small subset of the possible trajectories. These will typically be the more advantageous paths for the agent to pursue, with the less beneficial paths not represented at all. Indeed, the tree search is based on the expected free energy that favors policies that maximize information gain while realizing the prior preferences of the agent.

Additionally, the inference process can update the system's understanding of past contingencies on the basis of new observations. As a result, the system can obtain more refined information about previous decisions, perhaps re-evaluating the optimality of these past decisions. Because classic active inference represents a larger space of policies, this re-evaluation could apply to more policies.

We also know that humans engage in counterfactual reasoning (Rafetseder et al, 2013), which, in our planning context, could involve the entertainment and evaluation of alternative (non-selected) sequences of decisions. It may be that, because of the more exhaustive representation of possible trajectories, classic active inference can more efficiently engage in counterfactual reasoning. In contrast, branching-time active inference would require these alternative pasts to be generated "a fresh" for each counterfactual deliberation. In this sense, one might

argue that there is a trade off: branching-time active inference provides considerably more efficient planning to attain current goals, classic active inference provides a more exhaustive assessment of paths not taken.

3.2 The deep reward environment

In this section, we introduce a canonical example of the kind of environment in which BTAI outperforms standard active inference. This environment is called the deep reward environment because the agent needs to navigate a tree like graph, where the graph's nodes correspond to the states of the system, and the agent needs to look deep into the future to diferentiate the favourable path from the traps.

At the beginning of each trial, the agent is placed at the root of the tree that corresponds to the initial state (S_0) of the system. From the initial state, the agent can select m actions leading immendiately to an undesirable state, and n actions leading to seemingly pleasant states, for a total of $n+m$ actions. If one of the m undesirable actions is selected, then the agent will enter a bad path, in which (at each time step) $n+m$ actions are available, but all of them produce unpleasant observations. While these m undesirable actions that lead directly to terrible states should be straightforward to avoid for any reasonable agent, the n seemingly favourable actions present an additional challenge. Indeed, only one of those n actions will be beneficial to the agent in the long run, and all the others are long-term traps.

We let L_k with $k \in \{1, ..., n\}$ be the length of the k-th seemingly good path. Once the agent is engaged on the k-th path, there are still $n+m$ actions available, but only one of them keeps the agent on the right track. All the other actions will produce unpleasant observations, i.e., the agent will enter a bad path. This process will continue until the agent reaches the end of the k-th path, which is determined by the path's length L_k. If the k-th path was the longest of the n seemingly good paths, then the agent will from now on only receive pleasant observations independently of the action performed. If the k-th path was not the longest path, then independently of the action performed, the agent will receive painful observations, i.e., the trap is revealed.

To summarize, at the beginning of each trial, the agent is prompted with n seemingly good paths and m obviously bad paths. Only the longest of the seemingly pleasant paths will be beneficial in the long term, the other are traps, which will ultimately lead the agent to an undesirable state. Figure 3 illustrates this environment. Also in theory, this task does not have any terminal states, and the agent will keep taking actions forever. However, in practice, each trial is stopped after a fixed number of action-perception cycles.

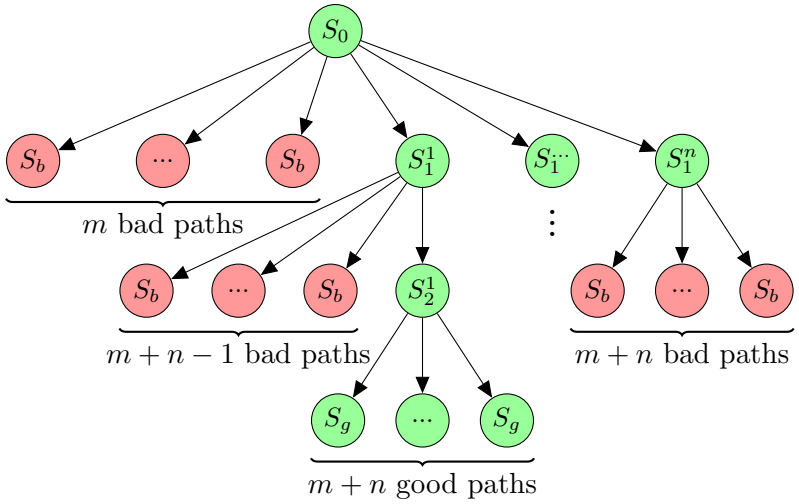


Figure 3: This figure illustrates a type of environment in which BTAI will outperform standard active inference. Typically, this corresponds to environments in which there are only a small number of good actions. In such environments, BTAI can safely discard a large part of the tree, and speed up the search without impacting performance. Note, S_0 represents the initial state, S_b represents a bad state, S_g represents a good state, and S_j^i is the j-th state of the i-th seemingly good path. The above picture assumes that the longest path (which is beneficial in the long-term) is the path starting with the state S_1^1. Its length (L_1) is equal to two because after performing two actions (i.e., the one leading to S_1^1 and the one leading to S_2^1), the agent is certain to receive pleasant observations. Importantly, any other (seemingly) good path starting with a state S_1^i with $i \in \{2, ..., n\}$ will turn out to be a trap. A trap is simply a state from which all actions lead to an undesirable state (S_b), e.g., S_1^n is a trap. Note, at each time point, the agent must pick from the $m+n$ possible actions, e.g, when reaching S_1^1 there is only one action keeping the agent on the right track, but all the other actions (i.e., $m+n-1$ actions) lead to a bad state.

3.2.1 The easy, medium and hard deep reward environment

In this section, we present three instances of the deep reward environment in increasing order of complexity (i.e., easy, medium, and hard). These instances will then be used to compare BTAI and (standard) active inference. To specify an instance completely, it is sufficient to provide the number of obviously detrimental actions (m), the number of seemingly good actions (n), and the length of the paths that follow from the seemingly good actions, i.e., L_k for $k \in \{1, ..., n\}$.

All three instances have five obviously detrimental actions ($m = 5$) and two seemingly good actions ($n = 2$). However, the lengths of the two good paths (i.e., L_1 and L_2) change from one instance to the other, and the reader is referred to Table 1 for a summary. In all the environments considered, $L_2 > L_1$, therefore the first path is a trap that will lead to an undesirable state, and the second path is the one that should be taken. Also, to identify that the first path is a trap, the agent must be able to plan at least $L_1 + 1$ steps ahead, since before that the two seemingly good paths are identical. Importantly, an agent trying to evaluate all possible policies $L_1 + 1$ steps into the future, will have to store and process: 343 policies for the easy instance, 16,807 policies for the medium instance, and 5,764,801 policies for the hard instance. We conclude this section with Figure 4 that illustrates the easy instance of the deep reward environment.

Environment	L_1	L_2
easy	2	3
medium	4	5
hard	7	9

Table 1: This table presents the three deep reward environments on which experiments will be run.

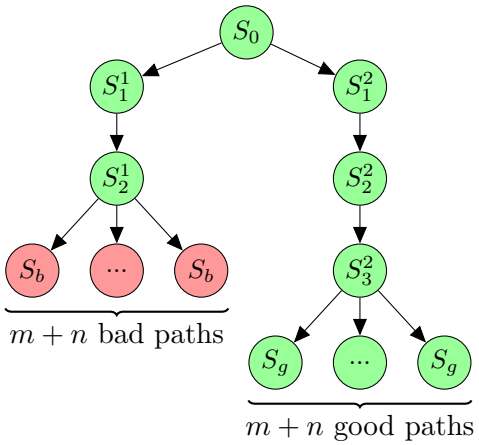


Figure 4: This figure illustrates the easy instance of the deep reward environment used to compare BTAI and AcI. It contains two seemingly good paths ($n = 2$): the first of length two ($L_1 = 2$) and the second of length three ($L_2 = 3$). Upon reaching the end of the first (and shortest) path, the agent can only reach undesirable states, i.e., the first path is a trap. In contrast, when reaching the end of the second (and longest) path, the agent can only reach pleasant states, i.e., the second path is beneficial in the long term. Importantly, the entire graph of the easy version contains more than 300 nodes, and is only partially represented. The exhaustive graph is obtained by adding undesirable states (S_b) until each node has $n + m$ children, e.g., S_0 has $m = 5$ unrepresented children and S_1^1 has six of them. Finally, the medium and hard versions of the deep reward environment can be obtained from the easy version by lengthening the two seemingly good paths.

3.3 BTAI vs active inference: Simulations

In this section, we compare BTAI and active inference on the three instances of the deep reward environment presented in Section 3.2.1. The Matlab code running an active inference agent was implemented by modifying the SPM demo called: `DEMO_MDP_maze.m`, and is publicly available on GitHub at the following URL: `https://github.com/ChampiB/Experiments_AI_TS`, in the file: `matlab/graph_navigation.m`.

Table 2 shows the result of our simulation in which a standard active inference agent is run on the three deep reward environments presented in Section 3.2.1. Since the behaviour of the simulation is deterministic, only one run was executed. If the agent successfully selects the longest path, we report $P(goal) = 1$, otherwise, we report $P(trap) = 1$. Lastly, the simulation was run on a standard laptop with 16GB of RAM, if the agent ran out of memory, then we simply report a "crash" in the table. As expected, the agent successfully solved the easy and medium environments, for which it was required to plan three and five steps ahead. However, for the hardest version, the agent was supposed to store and process more than five millions policies and the associated beliefs

over both: policies and hidden states. This is intractable using only 16GB of RAM and standard active inference runs out of memory because of the exponential (space) complexity class.

Environment	Policy size	P(goal)	P(trap)	Time (sec)
easy	3	1	0	14.79
medium	5	1	0	1177.05
hard	8	crash	crash	crash

Table 2: This table shows that the active inference agent was able to plan three and five time steps ahead to solve the easy and medium deep reward environments. However, because of the exponential space complexity, SPM runs out of memory when trying to plan eight time steps ahead to solve the hardest deep reward environment. The last column reports the time (in seconds) required for running one simulation of the graph environment using SPM.

The C++ code emulating BTAI can be found in the file `experiments/main.cpp` of the GitHub repository previously discussed (`ChampiB/Experiments_AI_TS`). The hyper-parameters used in the code are described in Appendix D. Since action selection in BTAI is stochastic, we ran 100 simulations. We report the probability of the agent selecting the longest path as: $P(goal) = \frac{\text{number of successes}}{100}$. Simulations where the agent failed to select the proper path are reported as: $P(trap) = \frac{\text{number of failures}}{100}$. We experimented with various numbers of planning iterations, starting with ten iterations and increasing this number by five until the agent was able to solve the task.

Table 3 shows the results obtained by BTAI on the three deep reward environments presented in Table 1, and the hyper-parameter values used in these simulations are reported in Appendix D. As expected, the agent successfully solved the three deep reward environments. Ten planning iterations were required for the easy and medium environments, and twenty for the hardest one. The ability of BTAI to find the best policy among more than five millions policies with only twenty planning iterations is explained by the sparcity of the deep reward environment, i.e., the vast majority of the policies are clearly detrimental to the agent. Note that this sparcity is characteristic of many complex tasks such as chess. For example, a chess player is frequently faced with (chess) positions where twenty to forty legal moves are available, but one move is almost forced, i.e., if not played, the player will almost surely lose the game.

Environment	Planning iterations	P(goal)	P(trap)	Time (sec)
easy	10	1	0	0.112 ± 0.008
medium	10	1	0	0.193 ± 0.007
hard	10	0.5	0.5	0.356 ± 0.020
	15	0.49	0.51	0.536 ± 0.052
	20	1	0	0.836 ± 0.075

Table 3: This table shows that BTAI was able to solve the three deep reward environments with at most 20 planning iterations. The reported time corresponds to the average runtime of one simulation, and the standard deviation is reported after the symbol $\pm$.

4. BTAI Empirical Intuition

In this section, we study the BTAI agent's behaviour through experiments highlighting its vulnerability to local minimum and ways to mitigate this issue. The goal is to gain some intuition about how the model behaves when: enabling deeper searches, providing better preferences, and using different kind of cost functions to guide the Monte Carlo tree search. The code of those experiments is available on GitHub at the following URL: `https://github.com/ChampiB/Experiments_AI_TS`, in the file: `experiments/main.cpp`.

4.1 The maze environment

This section presents the environment in which various simulations will be run. In this environment, the agent can be understood as a rat navigating a maze. Figure 5 illustrates the three mazes studied in the following sections. The agent can perform five actions, i.e., UP, DOWN, LEFT, RIGHT and IDLE. The goal is to reach the maze exit from the starting position of the agent. To do so, the agent must move from empty cells to empty cells avoiding walls. If the agent tries to move through a wall, the action becomes equivalent to IDLE. Finally, the observations made by the agent correspond to the Manhattan distance (with the ability to traverse walls) between its current position and the maze exit, i.e.,

$$M(x, y) = \sum_{i=1}^{N} |x_i - y_i|,$$

where $M(x, y)$ is the Manhattan distance between $x \in \mathbb{R}^N$ and $y \in \mathbb{R}^N$, x is the position of the agent, y the position of the exit, and in a 2d maze $N = 2$. Figure 5 (left) illustrates the Manhattan distance received on each cell of a simple maze. Taking maze (A) from Figure 5 as an example, if the agent stands on the exit (green square), the observation will be zero or equivalently using one-hot encoding[3] [1 0 0 0 0 0 0 0 0 0], and if the agent stands at the initial position (red square), the observation will be nine or equivalently [0 0 0 0 0 0 0 0 0 1].

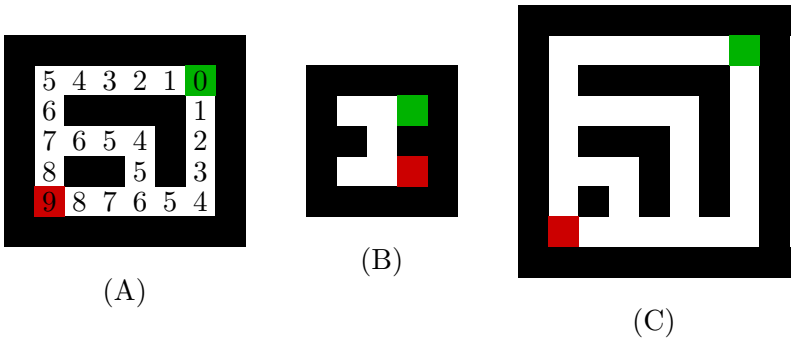


Figure 5: This figure illustrates the three mazes used to perform the experiments in the next sections. Black squares correspond to walls, green squares correspond to the maze exit and red squares correspond to the agent starting position. Finally, the numbers displayed on each cell of maze (A) correspond to the Manhattan distance between this cell and the exit.

3. A one-hot encoding of a number $n \in \{0, ..., N\}$ means representing n as a vector of size $N + 1$, where the n-th element is equal to one and all the other are set to zeros. In this paper, we assume a zero based indexing, i.e., the first element is at index zero.

4.2 Overcoming the challenge of local minima

In this section, we investigate the challenge of local minima and provide two ways of mitigating the issue: improving the prior preferences and using a deeper tree.

4.2.1 Prior preferences and local minimum

In this first experiment, the agent was asked to solve maze (B) from Figure 5, which has the property that the start location (red square) is a local minimum. Remember from Section 4.1 that the agent observes the Manhattan distance between its location and the maze exit. The Manhattan distance naturally creates local minima throughout the mazes, i.e., cells of the maze (apart from the exit) for which no adjacent cell has a lower distance to the exit. An example of such a local minimum is shown as a blue square in Figure 6. The presence of such a local minimum implies that a well behaved agent (i.e., an agent trying to get as close as possible to the exit) might get trapped in those cells for which no adjacent cell has a lower distance to the exit and thus fail to solve the task.

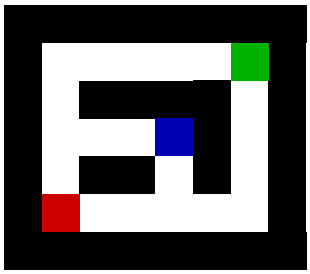


Figure 6: This figure illustrates the notion of local minimum (i.e., the blue cell) in the context of maze (A). Local minima correspond to cells (apart from the exit) for which no adjacent cell has a lower distance to the exit.

Next, we need to define the prior preferences of the agent. Our framework allows the modeller to define prior preferences over both future observations and future states. However, we start by assuming no preferences over the hidden states, i.e., $V(S_I)$ is uniform. We define the prior preferences over future observations as:

$$\boldsymbol{C}_O = \sigma\big(\gamma \boldsymbol{v}\big) \text{ with } \boldsymbol{v} = \big[\ |O|\ \ ...\ \ 2\ \ 1\ \big]^T$$

where $|O|$ is the number of possible observations (10 in maze (A) from Figure 5), γ is the precision of the prior preferences, and $\sigma(\cdot)$ is the softmax function. The above prior preferences will give high probability to cells close to the exit and will exhibit the local minimum behaviours previously mentioned.

Using these prior preferences, we ran 100 simulations in maze (B) from Figure 5. Each simulation was composed of a maximum of 20 action-perception cycles, and was interrupted when the agent reached the maze exit. Note, the results might vary from simulation to simulation, because the actions performed in the environment are sampled from $\sigma(-\omega \frac{g}{N})$, where $\sigma(\bullet)$ is a softmax function, ω is the precision of action selection, g is a vector whose elements

correspond to the cost of the root's children (i.e. the children of S_t) and N is a vector whose elements correspond to the number of visits of the root's children.

Table 4 reports the frequency at which the agent reaches the exit. The hyper-parameters values are reported in Appendix D. First, note that with 10 and 15 planning iterations, the agent was unable to leave the initial position (i.e., it is trapped in the local minimum). But as the number of planning iterations is increased, the agent becomes able to foresee the benefits of leaving the local minimum.

Planning iterations	P(exit)	P(local)	Time (sec)
10	0	1	0.701 ± 0.022
15	0	1	1.030 ± 0.070
20	1	0	0.233 ± 0.018

Table 4: This table presents the probability that the agent solves maze (B), and the probability of the agent being stuck into the local minimum. The reported time corresponds to the average runtime of one simulation, and the standard deviation is reported after the symbol ±. Importantly, when the agent reaches the exit of the maze the simulation is interrupted, i.e., the simulation contains less than 20 action-perception cycles. This explains why performing 20 planning iterations is faster (0.233 seconds), than performing 15 planning iterations (1.030 seconds), i.e., the simulations with 15 planning iterations (that fail to solve the maze) contain 20 action-perception cycles while the simulations with 20 planning iterations (that successfully solve the maze) contain less than 20 action-perception cycles.

4.2.2 Improving prior preference to avoid local minimum

In this second experiment, we modified the prior preferences of the agent to enable it to avoid local minima. We first change the cost function from the expected free energy $g_I^{classic}$ to the pure cost g_I^{pcost}, which allows us to set nontrivial preferences over states (in the previous section, these were set to uniform). Specifically, the prior preferences over hidden states will be of the form:

$$\boldsymbol{C}_S = \sigma\big(\gamma \boldsymbol{w}\big),$$

where γ is the precision over prior preferences, and $\boldsymbol{w}$ is set according to Figure 7. Finally, the prior preferences over future observations remain the same as in the previous section, and once again the hyper-parameters values are reported in Appendix D.

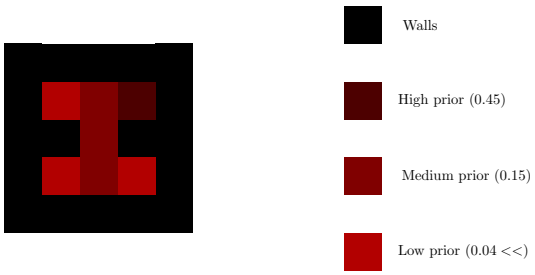


Figure 7: This figure illustrates the new prior preferences of the agent over the future states. Black squares correspond to walls, the darkest red corresponds to high prior preferences (really enjoyable states), the brightest red corresponds to low prior preferences (annoying states) and the last kind of red corresponds to medium prior preferences (boring states).

Tables 5 and 6 summarize the results of the experiments with and without the use of prior preferences over hidden states, respectively. As expected better prior preferences lead to better performance when less planning iterations are performed. Specifying prior preferences over hidden states requires the modeller to bring additional knowledge to the agent, and might not always be possible. However, when such knowledge is available it can improve the agent's performance. This illustrates the value of the BTAI approach, which enables preferences to be specified for observations, as does active inference, as well as for states.

Planning iterations	P(global)	P(local)	Time (sec)
10	0	1	0.683 ± 0.024
15	0	1	0.983 ± 0.030
20	1	0	0.217 ± 0.002

Table 5: This table presents the probability that the agent solves maze (B), and the probability of the agent being stuck in the local minimum. In this table, the agent was not equipped with prior preferences over hidden states. The last column reports the (average) execution time required for running one simulation and the associated standard deviation.

Planning iterations	P(global)	P(local)	Time (sec)
10	0	1	0.749 ± 0.045
15	1	0	0.181 ± 0.018
20	1	0	0.288 ± 0.092

Table 6: This table presents the probability that the agent solves maze (B), and the probability of the agent being stuck in the local minimum. In this table, the agent was equipped with prior preferences over hidden states. The last column reports the (average) execution time required for running one simulation and the associated standard deviation.

4.3 Solving more mazes

Up to now, we focused on maze (B) from Figure 5 to demonstrate that both improving prior preferences and deepening the tree can help to mitigate the problem of local minima. In this section, we extend our analysis to mazes (A) and (C). Table 7 shows the performance of the BTAI agent in maze (A) when using $g_I^{classic}$ and g_I^{pcost} as cost function. When g_I^{pcost} was used as a cost function, the agent was only equipped with prior preferences over

observations (i.e., uniform preferences over hidden states). Table 8 shows the results of the same experiments but on maze (C). As usual the hyper-parameters values used for those simulations are given in Appendix D.

Tables 7 and 8 seem to indicate that both $g_I^{classic}$ and g_I^{pcost} perform similiarly on the maze environment, and require approximatly the same amount of time to be computed. The similiar performance of $g_I^{classic}$ and g_I^{pcost} may be surprising to the reader. Indeed, $g_I^{classic}$ contains an ambiguity terms, i.e., $\mathbb{E}_{Q(S_J)}[\mathrm{H}[P(O_J|S_J)]]$, which should be helping the agent. In contrast, g_I^{pcost} contains the risk over states with uniform prior preferences over states, i.e., $D_{\mathrm{KL}}\left[\, Q(S_J) || \, V(S_J)\right]$, which should not be helpful (because of the uniformity of the prior preferences).

However, in the maze envionment the ambiguity of the likelihood mapping $P(O_\tau|S_\tau)$ is identical for each possible hidden state S_τ. Indeed, each state corresponds to a cell, and each cell is at a fix Manhattan distance from the exit. Thus, each state generates with high probability the observation corresponding to the Manhattan distance between the state's cell and the exit; and generates with small probability any other observations. For example, the likelihood mapping of an imaginary maze could be defined as follow:

$$P(O_\tau|S_\tau) = \boldsymbol{A} = \begin{bmatrix} 0.05 & 0.05 & 0.9 \\ 0.05 & 0.9 & 0.05 \\ 0.9 & 0.05 & 0.05 \end{bmatrix},$$

where $P(O_\tau = i|S_\tau = j) = \boldsymbol{A}_{ij}$. Importantly, each column of $\boldsymbol{A}$ has the same entropy, therefore the agent does not care about which observation is made, i.e., they are all as ambiguous. This is why the ambiguity term is in fact not helpful in the maze environment, and why $g_I^{classic}$ and g_I^{pcost} produce similar performances.

Planning iterations	P(global)	P(local)	Time (sec) for g_I^{classic}	Time (sec) for g_I^{pcost}
10	1	0	0.310 ± 0.032	0.287 ± 0.022
15	1	0	0.423 ± 0.008	0.432 ± 0.011
20	1	0	0.567 ± 0.026	0.579 ± 0.023

Table 7: This table presents the probability that the agent solves maze (A) from Figure 5, and the probability of the agent falling into the local minimum. Both cost functions $g_I^{classic}$ and g_I^{pcost} lead to the above results in maze (A). The last two columns report the (average) execution time and the associated standard deviation of running one simulation with $g_I^{classic}$ and g_I^{pcost}, respectively.

Planning iterations	P(global)	P(local)	Time (sec) for g_I^{classic}	Time (sec) for g_I^{pcost}
10	1	0	0.498 ± 0.053	0.460 ± 0.019
15	1	0	0.696 ± 0.063	0,664 ± 0.075
20	1	0	0.920 ± 0.091	0.833 ± 0.038

Table 8: This table presents the probability that the agent solves maze (C), and the probability of the agent falling into the local minimum. Both cost functions $g_I^{classic}$ and g_I^{pcost} lead to the above results in maze (C). The last two columns report the (average) execution time and the associated standard deviation of running one simulation with $g_I^{classic}$ and g_I^{pcost}, respectively.

5. The frozen lake environment

In this section, we evaluate our agent on the frozen lake environmnent introduced by OpenAI (Brockman et al, 2016). The frozen lake environment can be represented as a 2D grid with r rows and c columns. Each cell in the grid is either a frozen surface that can support the agent's weight or a hole on which the agent cannot step without receiving a heavy penalty. One of the cells with a frozen surface contains a frisbee that the agent needs to recover, i.e., this cell is the goal state. For our purpose, each cell is associated with a number describing its location, and the agent observes only its location in the lake. The agent can perform four actions (i.e., UP, DOWN, LEFT, RIGHT) at any point in time. Actions that would lead the agent to leave the lake (through the external boundary), are equivalent to doing nothing and the agent does not move.

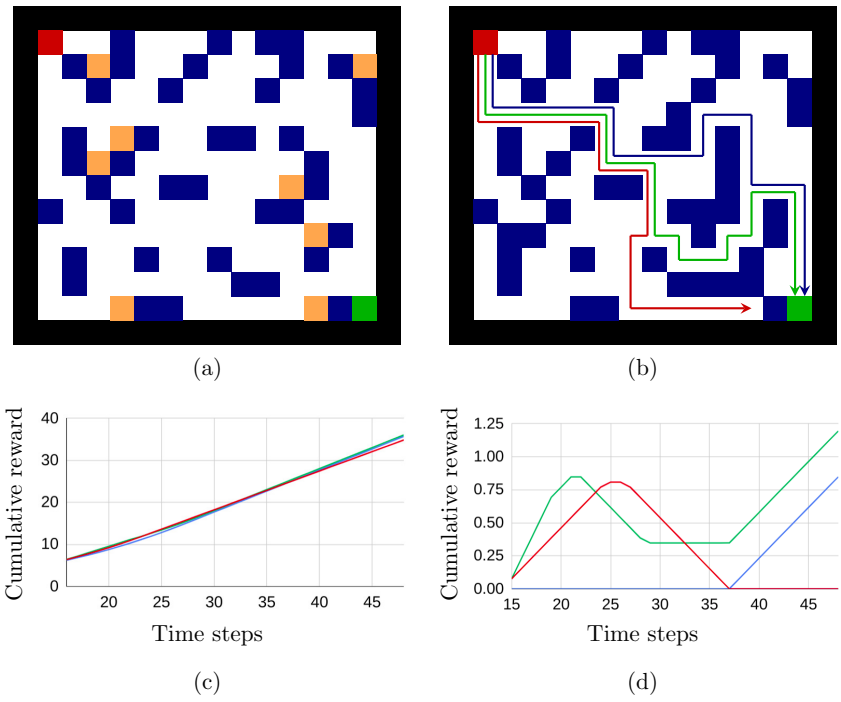


Figure 8: (a) and (b) illustrate the lakes used to perform the experiments of the present section. The black squares correspond to the external boundary of the lake, the green square corresponds to the frisbee location, the red squares correspond to the agent starting position, the orange squares correspond to local minima of the lake (not all local minima are represented), and the dark blue squares correspond to the holes in which the agent can fall if not careful. Note, these environments contain over 100 states, i.e., one for each cell within the external boundary. Finally, in (b) the green path corresponds to the path taken by the BTAI agent, the red path corresponds to the path selected by the POMCP agent (see the results in the main text), and the blue path corresponds to the shortest path connecting the starting position to the frisbee location. By the "shortest path", we mean the path that is passing through the smallest number of frozen surfaces without passing through a hole. (c) shows the cumulative reward (CR) received by the agent when following the green, red or blue path. The x-axis corresponds to the number of time steps, i.e., number of action-perception cycles, for which the agent follows the green, red or blue path. We see that all three paths have almost identical values. (d) shows the CR obtained along the green, red and blue paths minus the minimum cumulative reward (MCR) at each time step, where: $\text{MCR} = \min(\text{CR}_{\text{green}}, \text{CR}_{\text{red}}, \text{CR}_{\text{blue}})$.

In terms of the reward funtion, the agent receives a penalty of minus one each time it steps on a hole. Otherwise, the agent receives a reward between zero and one. This reward increases linearly as the agent gets closer to the frisbee location, where the distance between the agent and the frisbee is measured using the Manhattan distance as for the maze environment. Note, the reward received by the agent is maximum when the agent stands at the frisbee location, for which it receives a reward of one. Also in theory, this task does not have any terminal states, and the agent will keep taking actions forever. However, in practice, each trial is stopped after a fixed number of action-perception cycles. Figures 8(a) and 8(b) present the lakes in which the upcoming simulations have been ran. For reproducibility, we provide the values of the hyper-parameters used throughout this section in Appendix D.

5.1 BTAI on the frozen lake environment

Table 9 shows the results obtained by the BTAI agent on the lake of Figure 8(a). In short, the BTAI agent required twenty planning iterations before it was able to solve this task. Each simulation takes an average of 7.870 seconds of computational time, which correspond to approximatly $7.870/30 \approx 0.262$ seconds of thinking (i.e., inference, planning and action selection) per action-perception cycle.

Planning iterations	P(global)	P(local)	Time (sec)
10	0	1	6.991 ± 0.459
15	0	1	7.820 ± 0.577
20	1	0	7.870 ± 0.707

Table 9: This table presents the probability that the BTAI agent solves the lake of Figure 8(a), and the probability of the agent falling into a local minimum of the EFE. Where by "falling into a local minimum", we mean that the agent gets stuck into cells of the lake (apart from the exit) for which no adjacent cell represents a frozen surface that has a lower distance to the exit. The last column reports the execution time required for running one simulation and the associated standard deviation.

Table 10 shows the results obtained by the BTAI agent on the lake of Figure 8(b). In short, the BTAI agent requires fifty planning iterations to be able to solve this task. Each simulation takes an average of 19.187 seconds of computational time, which correspond to approximatly $19.187/30 \approx 0.639$ seconds of thinking (i.e., inference, planning and action selection) per action-perception cycle.

Planning iterations	P(global)	P(local)	Time (sec)
30	0	1	12.810 ± 1.071
40	0	1	15.589 ± 0.766
50	1	0	19.187 ± 1.317

Table 10: This table presents the probability that the BTAI agent solves the lake of Figure 8(b), and the probability of the agent falling into a local minimum of the EFE. Where by "falling into a local minimum", we mean that the agent gets stuck into cells of the lake (apart from the exit) for which no adjacent cell represents a frozen surface that has a lower distance to the exit. The last column reports the (average) execution time required for running one simulation, as well as the associated standard deviation.

5.2 POMCP on the frozen lake environment

In this section, we compare BTAI to the partially observable Monte Carlo planning (POMCP) algorithm introduced by Silver and Veness (2010). The code implementing the POMCP algorithm is available at the following URL: `https://github.com/ChampiB/POMCP`. Briefly, the POMCP agent performs MCTS (Silver et al, 2016; Browne et al, 2012; Schrittwieser et al, 2019) to select an action at each time step, and carries out inference using a particle filter (Doucet et al, 2009). Table 11 shows the results obtained by the POMCP agent on the lake of Figure 8(a). In short, the POMCP agent was able to reach the frisbee 97 % of the time when using one thousand planning iterations. At which point, each simulation takes an average of 40.444 seconds of computational time, which correspond to approximatly $40.444/30 \approx 1.348$ seconds of thinking (i.e., inference, planning and action selection) per action-perception cycle. This seems to indicate that BTAI is able to solve this first lake four times faster than the POMCP algorithm.

Planning iterations	P(global)	P(local)	Time (sec)
100	0.52	0.48	3.852 ± 0.227
500	0.89	0.11	20.550 ± 3.054
1000	0.97	0.03	40.444 ± 3.232
2000	0.93	0.07	83.156 ± 8.844

Table 11: This table presents the probability that the POMCP agent solves the lake of Figure 8(a), and the probability of the agent falling into a local maximum of the reward function. The last column of the above table reports the execution time required for running one simulation and the associated standard deviation. Importantly, this table can be compared with Table 9 that presents the performance of the BTAI agent on the same lake.

On the lake of Figure 8(b), the POMCP agent picks the red path, while the BTAI agent chooses the green path. As shown by Figure 8(c), even if BTAI reaches the goal state while POMCP does not, the cumulative reward obtained by both agents is almost identical. This means that both agents collect a similar amount of reward.

Interestingly, the approach receiving the largest amount of reward depends on the number of time steps in each simulation, i.e., the length of each episode. Figure 8(d) illustrates when BTAI is receiving more rewards than the POMCP algorithm, and when the opposite is true. To sum up, if a simulation is composed of between one and fifteen time step(s), both approaches are equivalent. If a simulation contains between sixteen and twenty-three action-perception cycles, BTAI will accumulate more rewards than the POMCP algorithm. If the simulation has between twenty-four and thirty-two time steps, then the POMCP agent will accumulate more rewards than BTAI. Lastly, if the simulation contains more than twenty-three action-perception cycles, BTAI will accumulate more rewards than the POMCP agent. Thus, in the long run, the POMCP algorithm selects a reasonable but slightly suboptimal path. This might be due to the small difference of cumulated reward obtained along the optimal path and the path taken by the POMCP algorithm. Also, this may be worsened both by the large number of time

steps required before to see any difference in accumulated reward between those two paths, and the variance of the MCTS algorithm (Veness et al, 2011).

Note, the blue path in Figure 8(b) is the shortest path connecting the starting position to the goal state, but is never optimal in terms of cumulative reward. This is because the blue path makes a detour through an area of the lake with low reward, while the green path makes a longer detour but passes through an area with higher rewards. Finally, if the reward received by the agent upon reaching the frisbee (i.e., green square) is increased sufficiently, then the POMCP agent gains incentive to cross the hole separating it from the frisbee, i.e., POMCP will accept a large penalty for an even greater reward.

6. The dSprites environment

The dSprites environment is based on the dSprites dataset (Matthey et al, 2017) initially designed for analysing the latent representation learned by variational auto-encoders (Doersch, 2016). The dSprites dataset is composed of images of squares, ellipses and hearts. Each image contains one shape (square, ellipse or heart) with its own size, orientation, and (X, Y) position. In the dSprites environment, the agent is able to move those shapes around by performing four actions (i.e., UP, DOWN, LEFT, RIGHT). To make planning tractable, the action selected by the agent is executed eight times in the environment before the beginning of the next action-perception cycle, i.e., the X or Y position is increased or decreased by eight between time step t and $t + 1$. The goal of the agent is to move all squares towards the bottom-left corner of the image and all ellipses and hearts towards the bottom-right corner of the image, c.f. Figure 9.

Since, BTAI is a tabular model whose likelihood $P(O_\tau|S_\tau)$ and transition $P(S_{\tau+1}|S_\tau, U_\tau)$ mappings are represented using matrices, the agent does not directly take images as inputs. Instead, the metadata of the dSprites dataset is used to specify the state space. In particular, the agent observes the type of shape (i.e., square, ellipse, or heart), as well as a coarse-grained version of the shape's true position. Importantly, the original images are composed of 32 possible values for both the X and Y positions of the shapes. A coarse-grained representation with a granularity of two means that the agent is only able to perceive 16×16 images, and thus, the positions at coordinate $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$ are indistinguishable. Figure 10 illustrates the coarse grained representation with a granularity of eight and the corresponding indices observed by the agent. Note that this modification of the observation space can be seen as a form of state aggregation (Ren and Krogh, 2002). Finally, as shown in Figure 10, the prior preferences of the agent are specified over an imaginary row below the dSprites image. This imaginary row ensures that the agent selects the action "down" when standing in the "appropriate corner", i.e., bottom-left corner for squares and bottom-right coner for ellipses and hearts.

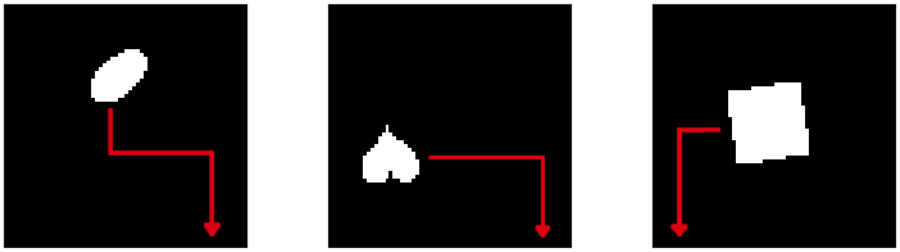
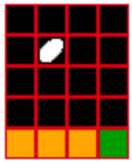


Figure 9: This figure illustrates the dSprites environment, in which the agent must move all squares towards the bottom-left corner of the image and all ellipses and hearts towards the bottom-right corner of the image. The red arrows show the behaviour expected from the agent.



0	5	10	15
1	6	11	16
2	7	12	17
3	8	13	18
4	9	14	19

□

20	25	30	35
21	26	31	36
22	27	32	37
23	28	33	38
24	29	34	39

♡

40	45	50	55
41	46	51	56
42	47	52	57
43	48	53	58
44	49	54	59

⬭

Figure 10: This figure illustrates the observations made by the agent when using a coarse-grained representation with a granularity of eight on the input image. On the left, one can see an image from the dSprites dataset and a grid containing red squares of 8×8 pixels. Any positions in those 8×8 squares are indistinguishable from the perspective of the agent. Also, the bottom most row is an imaginary row used to specify the prior preferences of the agent, i.e. the green square is the goal state and the orange squares correspond to undesirable states. Finally, the three tables on the right contain the indices observed by the agent for each type of shape at each possible position.

The evaluation of the agent's performance is based on the reward obtained by the agent. Briefly, the agent receives a reward of -1, if it never enters the imaginary row or if it does so at the antipode of the appropriate corner. As the agent enters the imaginary row closer and closer to the appropriate corner, its reward increases until reaching a maximum of 1. The percentage of the task solved (i.e., the evaluation metric) is calculated as follows:

$$P(\text{solved}) = \frac{\text{total rewards} + \text{number of runs}}{2.0 \times \text{number of runs}}.$$

Intuitively, the numerator shifts the rewards so that they are bounded between zero and two, and the denominator renormalises the reward to give a score between zero and one. A score of zero therefore corresponds to an agent always failing to enter the imaginary row or doing so at the antipode of the appropriate corner. In contrast, a score of one corresponds to an agent always entering the imaginary row through the appropriate corner.

6.1 BTAI on the dSprites environment

In this section, we evaluate BTAI on the dSprites environment. The hyper-parameters used in this section are presented in Appendix D. Briefly, the agent is able to solve 88.5% of the task when using a granularity of eight,

c.f. Table 12. To understand why BTAI was not able to solve the task with 100% accuracy, let us consider the example of an ellipse at position $(24, 31)$. With a granularity of eight, the agent perceives that the ellipse is in the bottom-right corner of the image, i.e., in the red square just above the goal state in Figure 10. From the agent's perspective, it is thus optimal to pick the action "down" to reach the goal state. However, in reality, the agent will not reveive the maximum reward because its true X position is 24 instead of the optimal X position of 31.

As shown in Table 13, we can improve the agent's perfomance, by using a granularity of four. This allows the agent to differentiate between a larger number of (X, Y) positions, i.e., it reduces the size of the red square in Figure 10. With this setting, the agent is able to solve 96.9% of the task. However, when decreasing the granularity, the number of states goes up, and so does the width and height of the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices. As an effect, more memory and computational time is required for the inference and planning process. This highlights a trade-off between the agent's performance and the amount of memory and time required. Indeed, a smaller granularity leads to better performance, but requires more time and memory.

Planning iterations	P(solved)	Time (sec)
10	0.813	0.859 ± 0.868
25	0.846	0.862 ± 0.958
50	0.885	1.286 ± 1.261

Table 12: This table presents the percentage of the dSprites environment solved by the BTAI agent when using a granularity of eight, c.f. Figure 10. The last column reports the average execution time required of one simulation and the associated standard deviation.

Planning iterations	P(solved)	Time (sec)
10	0.859	3.957 ± 4.027
25	0.933	3.711 ± 4.625
50	0.969	5.107 ± 5.337

Table 13: This table presents the percentage of the dSprites environment solved by the BTAI agent when using a granularity of four. In this setting, there are $9 \times 8 \times 3 = 216$ states. The last column reports the average execution time required of one simulation and the associated standard deviation.

6.2 Fountas et al approach on the dSprites environment

In this section, we experiment with the approach of Fountas et al (2020). The code used in this section is available on Github at the following URL: `https://github.com/ChampiB/deep-active-inference-mc`. First, we trained the agent for around two days on a Nvidia Tesla P100 GPU. After the training process, we ran 100 simulations on the original dSprites environment with both CPU and GPU. Table 14 reports the percentage of the task solved and the average time required for running a trial. Running the CPU simulations took on average 17.811 seconds per simulation. This is around three times longer than the GPU counterpart, which required an average of 5.467 seconds per simulation. Fountas' agent was able to solve up to 84.1 % of the task, which is less than the 96.9 % achieved by the BTAI agent in the previous section.

However, it is important to acknowledge the differences between the present paper and Fountas et al (2020), as well as the differences between the two environments on which those approaches have been evaluated. First, our approach is not equipped with deep neural networks, and is therefore unable to deal with images as input. Additionally, our agent was not asked to learn the environment's dynamics, instead, our agent was provided with a model of the environment since we are focusing on planning. In contrast, the agent of Fountas et al (2020) was able to successfully learn the environment's dynamics directly from images and then do the planning.

To conclude, our approach was able to solve 96.9 % of a simplified version of the dSprites environment, and the agent of Fountas et al (2020) was able to solve 84.1 % of the original dSprites environment. Additionally, our approach was provided with the environment's dynamics, while the agent of Fountas et al (2020) had to learn it, which took around two days on a Nvidia Tesla P100 GPU. Another, important trade-off is between interpretability and scalability. Indeed, the tabular representation of the likelihood and transition mappings makes the BTAI agent very intuitive and easy to understand. However, this tabular representation is also the main bottleneck blocking BTAI from solving image based environments. Similarly, the deep neural networks used by Fountas et al (2020) make their approach highly scalable, but also reduce the interpretability of the approach.

Computation type	P(solved)	Time (sec)
CPU	0.798	17.811 ± 19.143
GPU	0.841	5.467 ± 5.706

Table 14: This table presents the percentage of the original dSprites environment solved by the approach of Fountas et al (2020). The last column reports the average execution time required of one simulation and the associated standard deviation. Importantly, this table can be compared with Table 13 that presents the performance of the BTAI agent on a simplified version of the dSprites environment.

7. Conclusion and future works

In this paper, we provided an empirical study of branching time active inference (BTAI), where the name takes inspiration from branching-time theories of concurrent and distributed systems in computer science (Glabbeek, 1990; van Glabbeek, 1993; Bowman, 2005), and planning was cast as (temporal) structure learning. Simply put, the generative model is dynamically expanded and each expansion leads to the exploration of new policy fragments. The expansions are guided by the expected free energy, which provides a trade off between exploration and exploitation. Importantly, this approach is composed of not two, but three major distributions. The first is the prior distribution (or generative model) that encodes the agent's beliefs before performing any observations. The second is the posterior (or variational) distribution encoding the updated beliefs of the agent after performing some observations. And the third is a target distribution over future states and observations that encodes the prior preferences of the agent, i.e., a generalization of the $\boldsymbol{C}$ matrix in the standard formulation of active inference

proposed by Friston et al (2016a). An important advantage of this generalization is that it allows the specification of prior preferences over both future observations and future states at the same time.

We compared BTAI and standard active inference theoretically by studying its space and time complexity class. This study highlights that our method should perform better than the standard model used in active inference when the task can be solved by expanding the tree only a small number of times with respect to an exhaustive search. Second, we compared BTAI to active inference empirically within the deep reward environment. Those simulations suggest that BTAI is able to solve problems for which a standard active inference agent would run out of memory. Interestingly, active inference offers an Occam's window (Da Costa et al, 2020) for policy pruning, i.e., a policy is prunned if its posterior probability is very low w.r.t. the current best policy. This approach provides a way to reduce the amount of space used by active inference, since the policies with low probability and their associated beliefs over states can be discarded. However, a direct application of Occam's window will not solve the exponential time complexity class because the posterior probability of all policies still needs to be evaluated. It seems that a new AcI-based algorithm would be required to use the potential of Occam's window. As elaborated upon in Section 3.1, one might argue that there is a trade-off between banching-time active inference, which provides considerably more efficient planning to attain current goals, and classic active inference which provides a more exhaustive assessment of paths not taken. This might enable active inference to more exhaustively reflect counter-factuals and reasoning based upon them.

Also, BTAI was studied (experimentally) in the context of a maze solving task and we showed that when the heuristic used to create the prior preferences is not perfect, the agent becomes vulnerable to local minima. In other words, the agent might be attracted by a part of the maze that has low cost but does not allow it to solve the task. Then, we demonstrated empirically that improving the prior preferences of the agent by specifying a good prior over future hidden states and deepening the tree search, helped to mitigate this issue.

Moreover, BTAI was compared to the POMCP algorithm (Silver and Veness, 2010) on the frozen lake environment. This comparison was based upon two lakes each having their own topology. In terms of performance, both approaches successfully solved the simplest lake. On the hardest lake, BTAI and the POMCP algorithm received a similar amount of reward. Also, we described when BTAI receives more rewards than the POMCP agent, and when the opposite is true.

Additionally, BTAI was compared to the approach of Fountas et al (2020) on the dSprites dataset. The experiments show that our approach was able to solve 96.9 % of a simplified version of the dSprites environment, and the agent of Fountas et al (2020) was able to solve 84.1 % of the original dSprites environment. However, our approach was provided with the environment's dynamics, while the agent of Fountas et al (2020) had to learn it, which took around two days on a Nvidia Tesla P100 GPU. Another, important trade-off is between interpretability and scalability. Indeed, the tabular representation of the likelihood and transition mappings makes the BTAI

agent very intuitive and easy to understand. Unfortunatly, this tabular representation is also the main bottleneck blocking BTAI from solving image based environments. Similarly, the deep neural networks used by Fountas et al (2020) make their approach highly scalable, but reduce the interpretability of this approach.

The present paper could lead to a large number of future research directions. One could for example add the ability of the agent to learn the transition matrices $\boldsymbol{B}$ as well as the likelihood matrix $\boldsymbol{A}$ and the vector of initial states $\boldsymbol{D}$. This can be done in at least two ways. The first is to add Dirichlet priors over those matrices/vectors and the second would be to use neural networks as function approximators. The second option will lead to a deep active inference agent (Sancaktar and Lanillos, 2020; Millidge, 2020) equiped with tree search that could be directly compared to the method of Fountas et al (2020). Including deep neural networks in the framework will also open the door to direct comparison with the deep reinforcement learning literature (Haarnoja et al, 2018; Mnih et al, 2013; van Hasselt et al, 2015; Lample and Chaplot, 2017; Silver et al, 2016). Those comparisons will enable the study of the impact of the epistemic terms when the agent is composed of deep neural networks.

Another, important direction of research would be to learn the prior preferences of the agent (Sajid et al, 2021). Those preferences are encoded by the vector $\boldsymbol{C}$, and could be learned by incorporating a Dirichlet prior over $\boldsymbol{C}$. Also, the incorporation of this Dirichlet prior leads to an augmented EFE that could be compared with the standard formulation of the EFE.

Moreover, while the present paper is based on standard active inference that advocates that actions maximize both reward and information gain, it would be interresting to design a version of BTAI based on meta-control (Marković et al, 2021). Meta-control is a hierarchical model where higher-level hidden states constrain decision making at lower levels. Interestingly, Marković et al (2021) argue that it may be beneficial for the agent to switch on and off its exploration tendency based on the current context.

Another direction of research will be to set up behavioural experiments to try to determine which kind of planning is used by the brain. This could simply be done by looking at the time required by a human to solve various mazes and compare it with both the classic model and the tree search alternative. Finally, one could also set up a hierarchical model of action and compare it to the tree search algorithm presented here. One could also evaluate the plausibility of a hierarchical model of action by running behavioural experiments on humans.

Finally, a completely different direction will be to focus on the integration of memory. At the moment, when a new action is performed in the environment and a new observation is recieved from it, all the branches in the tree are prunned and a new temporal slice (i.e. a new state, action and observation triple) is added to the POMDP. In other words, the integration function simply records the past. This exact recording of the past is very unlikely to really happen in the brain. Therefore, one might simply ask what to do with this currently ever growing record of the past. This would certainly lead to the notion of an active inference agent equipped with episodic memory (Botvinick et al, 2019).

Acknowledgments

We would like to thank the reviewers for their valuable feedback, which greatly improved the quality of the present paper.

Appendix A: The theoretical approach of this paper.

This appendix describes the generative model, the variational distribution and the update equations used throughout this paper. For full details of vocabulary and notation the reader is referred to Champion et al (2021a).

The generative model can be understood as a fixed part modelling the past and present, and an expandable part modelling the future. The past and present is represented as a sequence of hidden states, where the transition between any two consecutive states depends on the action performed and is modelled using the 3-tensor $\boldsymbol{B}$. The generation of an observation is modelled by the matrix $\boldsymbol{A}$, and the prior over the initial hidden state as well as the prior over the various actions are modelled using vectors, i.e., $\boldsymbol{D}$ and $\boldsymbol{\Theta}_\tau$, respectively.

Concerning the second part of the model (i.e., the one modelling the future), the transition between consecutive states in the future is defined using the 2-sub-tensor $\boldsymbol{B}(\bullet, \bullet, I_{\text{last}})$, which is the matrix corresponding to the last action performed to reach the node S_I. The generation of future observations from future hidden states is identical to the one used for the past and present.

For the sake of simplicity, we assume that the tensors $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{D}$ and $\boldsymbol{\Theta}_\tau$ are given to the agent, which means that the agent knows the dynamics of the environment (c.f., Table 15 for additional information about those tensors). Practically, this means that the generative model does not have Dirichlet priors over those tensors. Furthermore, we follow Parr and Friston (2018), by viewing future observations as latent random variables. The formal definition of the generative model, which encodes our prior knowledge of the task, is given by:

$$\begin{aligned} P(O_{0:t}, S_{0:t}, U_{0:t-1}, O_{\mathbb{I}}, S_{\mathbb{I}}) =& P(S_0) \prod_{\tau=0}^{t-1} P(U_\tau) \prod_{\tau=0}^{t} P(O_\tau|S_\tau) \prod_{\tau=1}^{t} P(S_\tau|S_{\tau-1}, U_{\tau-1}) \\ & \prod_{I \in \mathbb{I}} P(O_I|S_I) P(S_I|S_{I\setminus\text{last}}) \end{aligned}$$

where $\mathbb{I}$ is the set of all non-empty multi-indexes already expanded, and $S_{I\setminus\text{last}}$ is the parent of S_I. Additionally, we need to define the individual factors:

$$\begin{aligned} &P(S_0) = \text{Cat}(\boldsymbol{D}) && P(U_\tau) = \text{Cat}(\boldsymbol{\Theta}_\tau) \\ &P(O_\tau|S_\tau) = \text{Cat}(\boldsymbol{A}) && P(O_I|S_I) = \text{Cat}(\boldsymbol{A}) \\ &P(S_\tau|S_{\tau-1}, U_{\tau-1}) = \text{Cat}(\boldsymbol{B}) && P(S_I|S_{I\setminus\text{last}}) = \text{Cat}(\boldsymbol{B}[I_{\text{last}}]). \end{aligned}$$

where I_{last} is the last index of the multi-index I, i.e., the last action that led to S_I, and $\boldsymbol{B}[I_{\text{last}}] = \boldsymbol{B}(\bullet, \bullet, I_{\text{last}})$ is the matrix corresponding to I_{last}. We now turn to the definition of the variational posterior. Under the mean-field approximation:

$$Q(S_{0:t}, U_{0:t-1}, O_{\mathbb{I}}, S_{\mathbb{I}}) = \prod_{\tau=0}^{t-1} Q(U_\tau) \prod_{\tau=0}^{t} Q(S_\tau) \prod_{I \in \mathbb{I}} Q(O_I) Q(S_I)$$

where the individual factors are defined as:

$$Q(S_\tau) = \text{Cat}(\hat{\boldsymbol{D}}_\tau) \qquad Q(U_\tau) = \text{Cat}(\hat{\boldsymbol{\Theta}}_\tau)$$

$$Q(O_I) = \text{Cat}(\hat{\boldsymbol{E}}_I) \qquad Q(S_I) = \text{Cat}(\hat{\boldsymbol{D}}_I)$$

Lastly, we follow Millidge et al (2021) in assuming that the agent aims to minimise the KL divergence between the variational posterior and a desired (target) distribution. Therefore, our framework allows for the specification of prior preferences over both future hidden states and future observations:

$$V(O_{\mathbb{I}}, S_{\mathbb{I}}) = \prod_{I \in \mathbb{I}} V(O_I) V(S_I)$$

where the individual factors are defined as:

$$V(O_I) = \text{Cat}(\boldsymbol{C}_O), \qquad V(S_I) = \text{Cat}(\boldsymbol{C}_S).$$

Importantly, $\boldsymbol{C}_O$ and $\boldsymbol{C}_S$ play the role of the vector $\boldsymbol{C}$ in the active inference model (Friston et al, 2016a), i.e., they specify which observations and hidden states are rewarding. To sum up, this framework is defined using three distributions: the prior defines the agent's beliefs before performing any observation, the posterior is an updated version of the prior that takes into account the observation made by the agent, and the target (desired) distribution encodes the agent's prior preferences in terms of future observations and hidden states.

Notation	Meaning
T, t	The time horizon and the current time step
$O_{i:j}$, $S_{i:j}$, $U_{i:j}$	The set of observations, states and actions between time step i and j (inclusive)
$\boldsymbol{A}$	The matrix defining the mapping from states to observations
$\boldsymbol{B}/\hat{\boldsymbol{D}}_\tau$	The 3-tensor defining the mappings (a priori) between any two consecutive hidden states and the parameters of the posterior over S_τ
$\boldsymbol{D}/\hat{\boldsymbol{D}}_0$	The parameters of the prior/posterior over the initial hidden states
$\hat{\boldsymbol{D}}_I/\hat{\boldsymbol{E}}_I$	The parameters of the posterior over future states/observations
$\boldsymbol{C}_S/\boldsymbol{C}_O$	The parameters of the prior preferences over future states/observations
$\boldsymbol{\Theta}_\tau/\hat{\boldsymbol{\Theta}}_\tau$	The parameters of the prior/posterior over actions at time step τ
$\sigma(\bullet)$	The softmax function
$\text{Cat}(\bullet)$ and $\text{Dir}(\bullet)$	Categorical and Dirichlet distributions

Table 15: Branching time active inference notation

Finally, the update equations used in this paper rely on variational message passing as presented in (Champion et al, 2021b; Winn and Bishop, 2005) and are given by:

$$
\begin{aligned}
Q^*(S_\tau) = \sigma\Big(& [\tau = 0] \ln \boldsymbol{D} \; + \; [\tau \neq 0] \ln \boldsymbol{B} \odot [\hat{\boldsymbol{D}}_{\tau-1}, \hat{\boldsymbol{\Theta}}_{\tau-1}] \\
& + \ln \boldsymbol{A} \odot \boldsymbol{o}_\tau \\
& + [\tau = t] \sum_{J \in \text{ch}_t} \ln \boldsymbol{B}[J_{last}] \odot \hat{\boldsymbol{D}}_J \; + \; [\tau \neq t] \ln \boldsymbol{B} \odot [\hat{\boldsymbol{D}}_{\tau+1}, \hat{\boldsymbol{\Theta}}_\tau] \Big) \\
Q^*(U_\tau) = \sigma\big(& \ln \boldsymbol{\Theta}_\tau + \ln \boldsymbol{B} \odot [\hat{\boldsymbol{D}}_\tau, \hat{\boldsymbol{D}}_{\tau+1}] \big) \\
Q^*(O_I) = \sigma\big(& \ln \boldsymbol{A} \odot \hat{\boldsymbol{D}}_I \big) \\
Q^*(S_I) = \sigma\big(& \ln \boldsymbol{A} \odot \hat{\boldsymbol{E}}_I + \ln \boldsymbol{B}[I_{last}] \odot \hat{\boldsymbol{D}}_{I \setminus \text{last}} + \sum_{S_K \in \text{ch}_I} \ln \boldsymbol{B}[K_{last}] \odot \hat{\boldsymbol{D}}_K \big)
\end{aligned}
$$

where $\boldsymbol{o}_\tau$ is the observation made at time step τ, I_{last} is the last action of the sequence I, ch_t is the set of multi-indices corresponding to the children of the root node, and ch_I is the set of multi-indices corresponding to the children of S_I. For additional information about $\odot$, the reader is referred to Appendix C.

Appendix B: Derivation of g_J^{pcost}.

In this appendix, we provide a derivation of g_J^{pcost} from the Free Energy of the Expected Future (FEEF) introduced by Millidge et al (2021):

$$g_I^{feef} = D_{\mathrm{KL}}\left[\, Q(O_I, S_I) || \, V(O_I, S_I)\right],$$

by assuming the following factorizations for the variational posterior:

$$Q(O_I, S_I) = Q(O_I)Q(S_I),$$

and target distribution:

$$V(O_I, S_I) = V(O_I)V(S_I).$$

Starting from g_I^{feef}, we use the definition of the KL divergence, the linearity of the expectation, the log property $\ln(ab) = \ln(a) + \ln(b)$, and the two assumptions described above to get:

$$\begin{aligned}
g_I^{feef} &= D_{\mathrm{KL}}\left[\, Q(O_I, S_I) || \, V(O_I, S_I)\right] \\
&= D_{\mathrm{KL}}\left[\, Q(O_I)Q(S_I) || \, V(O_I)V(S_I)\right] && \text{(factorization assumptions)} \\
&= \mathbb{E}_{Q(O_I)Q(S_I)}\big[\ln Q(O_I)Q(S_I) - \ln V(O_I)V(S_I)\big] && \text{(KL divergence definition)} \\
&= \mathbb{E}_{Q(O_I)Q(S_I)}\big[\ln Q(O_I) - \ln V(O_I) + \ln Q(S_I) - \ln V(S_I)\big] && \text{(log property)} \\
&= \mathbb{E}_{Q(O_I)}\big[\ln Q(O_I) - \ln V(O_I)\big] + \mathbb{E}_{Q(S_I)}\big[\ln Q(S_I) - \ln V(S_I)\big] && \text{(linearity of expectation)} \\
&= D_{\mathrm{KL}}\left[\, Q(O_I) || \, V(O_I)\right] + D_{\mathrm{KL}}\left[\, Q(S_I) || \, V(S_I)\right] && \text{(KL divergence definition)} \\
&= g_J^{pcost}.
\end{aligned}$$

Appendix C: Generalized inner product

Generalized inner products: Given an N dimensional tensor W and $M = N - 1$ vectors V^i, the generalized inner product returns a vector Z obtained by performing a weighted average (with weighting coming from the vectors) over all but one dimension. In other words:

$$Z = W \odot \left[V^1, ..., V^M\right] \Leftrightarrow Z(x_j) = \sum_{\substack{x_1 \in \{1,...,|V^1|\} \\ ... \\ x_M \in \{1,...,|V^M|\}}} V^1_{x_1} \times ... \times W(x_1, ..., x_j, ..., x_M) \times ... \times V^M_{x_M}$$

$$\forall x_j \in \{1, ..., |Z|\},$$

where $|Z|$ denotes the number of elements in Z, and the large summand is over all x_r for $r \in \{1, ..., M\} \setminus \{j\}$, i.e., excluding j. Also, note that if $|W|_{V^i}$ $\forall i \in \{1, ..., M\}$ is the number of elements in the dimension corresponding to V^i, then for $W \odot \left[V^1, ..., V^M\right]$ to be properly defined, we must have $|W|_{V^i} = |V^i|$ $\forall i \in \{1, ..., M\}$ where $|V^i|$ is the number of elements in V^i. Figure 11 illustrates the generalized inner product for $N = 3$.

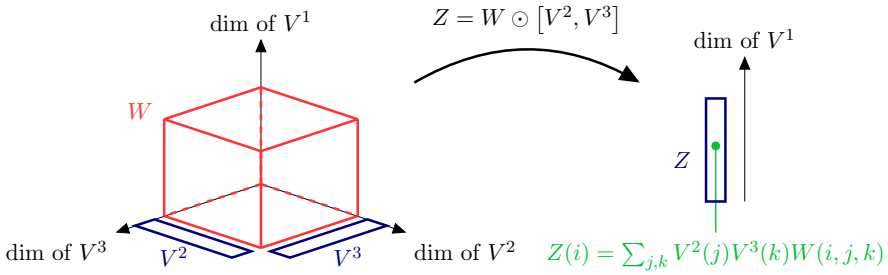


Figure 11: This figure illustrates the generalized inner product $Z = W \odot \left[V^2, V^3\right]$, where W is a cube of values illustrated in red with typical element $W(i, j, k)$. Also, the vectors Z and V^i $\forall i \in \{2, 3\}$ are drawn in blue along the dimension of the cube they correspond to.

Naming of the dimensions: Importantly, we should imagine that each side of W has a name, e.g., if W is a 3x2 matrix, then the i-th dimension of W could be named: "the dimension of V_i". This enables us to write: $Z^1 = W \odot V^1$ and $Z^2 = W \odot V^2$, where Z^1 is a 1x2 matrix (i.e., a vector with two elements) and Z^2 is a 3x1 matrix (i.e., a vector with three elements). The operator $\odot$ knows (thanks to the dimension name) that $W \odot V^1$ takes the weighted average w.r.t "the dimension of V_1", while $W \odot V^2$ must take the weighted average over "the dimension of V_2".

In the context of active inference, the matrix $\boldsymbol{A}$ has two dimensions that we could call "the observation dimension" (i.e., row-wise) and "the state dimension" (i.e., column-wise). Trivially, $\boldsymbol{A} \odot \boldsymbol{o}_\tau$ will then correspond to the average of $\boldsymbol{A}$ along the observation dimension and $\boldsymbol{A} \odot \hat{\boldsymbol{D}}_\tau$ will correspond to the average of $\boldsymbol{A}$ along the state dimension.

Appendix D: Hyper-parameters used during the simulations

Lists of hyper-parameters: Table 16 describes the role of the hyper-parameters of the BTAI simulation.

Name	Description
NB_SIMULATIONS	The number of simulations run during the experiment.
NB_ACTION_PERCEPTION_CYCLES	The maximum number of actions in each simulation, after which the simulation is terminated.
NB_PLANNING_STEPS	The number of planning iterations performed by the agent.
EXPLORATION_CONSTANT	The exploration constant of the UCT criterion.
PRECISION_PRIOR_PREFERENCES	The precision of the prior preferences, i.e., γ in $\boldsymbol{C}_O = \sigma(\gamma \boldsymbol{v})$, where $\boldsymbol{v}$ is a vector quantifying the preferences of the agent.
PRECISION_ACTION_SELECTION	The precision of the distribution used for action selection, i.e., ω in $\sigma(-\omega \frac{g}{N})$ where g is a vector whose elements correspond to the cost of the root's children (i.e. the children of S_t) and N is a vector whose elements correspond to the number of visits of the root's children.
EVALUATION_TYPE	The type of cost used to evaluate the node during the tree search, i.e., G_I^{classic} reported as EFE or G_I^{pcost} reported as DOUBLE_KL.

Table 16: This table describes the hyper-parameters of the BTAI simulation.

Table 17 describes the role of the hyper-parameters of the POMCP simulation.

Name	Description
NB_SIMULATIONS	The number of simulations run during the experiment.
NB_ACTION_PERCEPTION_CYCLES	The maximum number of actions in each simulation, after which the simulation is terminated.
TIMEOUT	The number of planning iterations performed by the agent.
EXP_CONST	The exploration constant of the UCT criterion.
GAMMA	The value of the discount factor.
NO_PARTICLES	The number of particles in the filter.

Table 17: This table describes the hyper-parameters of the POMCP simulation.

Hyper-parameters used by BTAI in section 3.3: Table 18 provides the value of each hyper-parameter used by BTAI in section 3.3.

Name	Value
NB_SIMULATIONS	100
NB_ACTION_PERCEPTION_CYCLES	20
NB_PLANNING_STEPS	10 or 15 or 20
EXPLORATION_CONSTANT	2.4
PRECISION_PRIOR_PREFERENCES	3
PRECISION_ACTION_SELECTION	100
EVALUATION_TYPE	EFE

Table 18: This table presents the value of each hyper-parameter used by BTAI in section 3.3.

Hyper-parameters used by BTAI in section 4.2.1: Table 19 provides the value of each hyper-parameter used by BTAI in section 4.2.1.

Name	Value
NB_SIMULATIONS	100
NB_ACTION_PERCEPTION_CYCLES	20
NB_PLANNING_STEPS	10 or 15 or 20
EXPLORATION_CONSTANT	2.4
PRECISION_PRIOR_PREFERENCES	2
PRECISION_ACTION_SELECTION	100
EVALUATION_TYPE	EFE

Table 19: This table presents the value of each hyper-parameter used by BTAI in section 4.2.1.

Hyper-parameters used by BTAI in section 4.2.2: Table 20 provides the value of each hyper-parameter used by BTAI in section 4.2.2.

Name	Value
NB_SIMULATIONS	100
NB_ACTION_PERCEPTION_CYCLES	20
NB_PLANNING_STEPS	10 or 15 or 20
EXPLORATION_CONSTANT	2.4
PRECISION_PRIOR_PREFERENCES	2
PRECISION_ACTION_SELECTION	100
EVALUATION_TYPE	DOUBLE_KL

Table 20: This table presents the value of each hyper-parameter used by BTAI in section 4.2.2.

Hyper-parameters used by BTAI in section 4.3: Table 21 provides the value of each hyper-parameter used by BTAI in section 4.3.

Name	Value
NB_SIMULATIONS	100
NB_ACTION_PERCEPTION_CYCLES	20
NB_PLANNING_STEPS	10 or 15 or 20
EXPLORATION_CONSTANT	2.4
PRECISION_PRIOR_PREFERENCES	2
PRECISION_ACTION_SELECTION	100
EVALUATION_TYPE	EFE or DOUBLE_KL

Table 21: This table presents the value of each hyper-parameter used by BTAI in section 4.3.

Hyper-parameters used by BTAI in section 5: Table 22 provides the value of each hyper-parameter used by BTAI in section 5.

Name	Value
NB_SIMULATIONS	100
NB_ACTION_PERCEPTION_CYCLES	30
NB_PLANNING_STEPS	10, 15, 20, 30, 40 or 50
EXPLORATION_CONSTANT	2.4
PRECISION_PRIOR_PREFERENCES	2
PRECISION_ACTION_SELECTION	100
EVALUATION_TYPE	EFE

Table 22: This table presents the value of each hyper-parameter used by BTAI in section 5. Note, the number of action-perception cycles has been increased from 20 to 30, because the agent cannot possibly solve the task with 20 actions (the lake is too large).

Hyper-parameters used by the POMCP algorithm in section 5: Table 23 provides the value of each hyper-parameter used by the POMCP algorithm in section 5.

Name	Value
NB_SIMULATIONS	100
NB_ACTION_PERCEPTION_CYCLES	30
TIMEOUT	100, 500, 1000 or 2000
EXP_CONST	3
GAMMA	0.9
NO_PARTICLES	100

Table 23: This table presents the value of each hyper-parameter used by the POMCP algorithm in section 5.

Hyper-parameters used by BTAI in section 6.1: Table 24 provides the value of each hyper-parameter used by BTAI in section 6.1. Also, note that the granularity of the coarse-grained representation was set to four or eight.

Name	Value
NB_SIMULATIONS	100
NB_ACTION_PERCEPTION_CYCLES	30
NB_PLANNING_STEPS	10, 25 or 50
EXPLORATION_CONSTANT	2.4
PRECISION_PRIOR_PREFERENCES	2
PRECISION_ACTION_SELECTION	100
EVALUATION_TYPE	EFE

Table 24: This table presents the value of each hyper-parameter used by BTAI in section 6.1.

References

Botvinick M, Toussaint M (2012) Planning as inference. Trends in Cognitive Sciences 16(10):485 – 488, DOI https://doi.org/10.1016/j.tics.2012.08.006

Botvinick M, Ritter S, Wang JX, Kurth-Nelson Z, Blundell C, Hassabis D (2019) Reinforcement learning, fast and slow. Trends in Cognitive Sciences 23(5):408 – 422, DOI https://doi.org/10.1016/j.tics.2019.02.006, URL http://www.sciencedirect.com/science/article/pii/S1364661319300610

Bowman H (2005) Concurrency Theory: Calculi an Automata for Modelling Untimed and Timed Concurrent Systems. Springer, Dordrecht, URL https://cds.cern.ch/record/1250124

Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W (2016) Openai gym. arXiv:1606.01540

Browne CB, Powley E, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S (2012) A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games 4(1):1–43

Butz MV, Bilkey D, Humaidan D, Knott A, Otte S (2019) Learning, planning, and control in a monolithic neural event inference architecture. Neural Networks 117:135–144, DOI https://doi.org/10.1016/j.neunet.2019.05.001, URL https://www.sciencedirect.com/science/article/pii/S0893608019301339

Catal, Ozan and Verbelen, Tim and Nauta, Johannes and De Boom, Cedric and Dhoedt, Bart (2020) Learning perception and planning with deep active inference. In: ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, pp 3952–3956, URL http://dx.doi.org/10.1109/ICASSP40776.2020.9054364

Champion T, Bowman H, Grześ M (2021a) Branching time active inference: the theory and its generality

Champion T, Grześ M, Bowman H (2021b) Realizing Active Inference in Variational Message Passing: The Outcome-Blind Certainty Seeker. Neural Computation pp 1–65, DOI 10.1162/neco_a_01422, URL https://doi.org/10.1162/neco_a_01422, https://direct.mit.edu/neco/article-pdf/doi/10.1162/neco_a_01422/1930278/neco_a_01422.pdf

Cox M, van de Laar T, de Vries B (2019) A factor graph approach to automated design of Bayesian signal processing algorithms. Int J Approx Reason 104:185–204, DOI 10.1016/j.ijar.2018.11.002, URL https://doi.org/10.1016/j.ijar.2018.11.002

Cullen M, Davey B, Friston KJ, Moran RJ (2018) Active Inference in OpenAI Gym: A Paradigm for Computational Investigations Into Psychiatric Illness. Biological Psychiatry: Cognitive Neuroscience and Neuroimaging 3(9):809 – 818, DOI https://doi.org/10.1016/j.bpsc.2018.06.010, URL http://www.sciencedirect.com/science/article/pii/S2451902218301617, computational Methods and Modeling in Psychiatry

Da Costa L, Parr T, Sajid N, Veselic S, Neacsu V, Friston K (2020) Active inference on discrete state-spaces: A synthesis. Journal of Mathematical Psychology 99:102,447, DOI https://doi.org/10.1016/j.jmp.2020.102447, URL https://www.sciencedirect.com/science/article/pii/S0022249620300857

Doersch C (2016) Tutorial on variational autoencoders. 1606.05908

Doucet A, Johansen AM, et al (2009) A tutorial on particle filtering and smoothing: Fifteen years later. Handbook of nonlinear filtering 12(656-704):3

FitzGerald THB, Dolan RJ, Friston K (2015) Dopamine, reward learning, and active inference. Frontiers in Computational Neuroscience 9:136, DOI 10.3389/fncom.2015.00136, URL https://www.frontiersin.org/article/10.3389/fncom.2015.00136

Forney GD (2001) Codes on graphs: normal realizations. IEEE Transactions on Information Theory 47(2):520–548

Fountas Z, Sajid N, Mediano PAM, Friston K (2020) Deep active inference agents using Monte-Carlo methods. arXiv e-prints arXiv:2006.04176, 2006.04176

Fox CW, Roberts SJ (2012) A tutorial on variational Bayesian inference. Artificial Intelligence Review 38(2):85–95, DOI 10.1007/s10462-011-9236-8, URL https://doi.org/10.1007/s10462-011-9236-8

Friston K (2010) The free-energy principle: a unified brain theory? Nature Reviews Neuroscience 11(2):127–138, DOI 10.1038/nrn2787, URL https://doi.org/10.1038/nrn2787

Friston K, FitzGerald T, Rigoli F, Schwartenbeck P, Doherty JO, Pezzulo G (2016a) Active inference and learning. Neuroscience & Biobehavioral Reviews 68:862 – 879, DOI https://doi.org/10.1016/j.neubiorev.2016.06.022

Friston K, Parr T, Zeidman P (2018) Bayesian model reduction. arXiv e-prints arXiv:1805.07092, 1805.07092

Friston K, Da Costa L, Hafner D, Hesp C, Parr T (2021) Sophisticated Inference. Neural Computation 33(3):713–763, DOI 10.1162/neco_a_01351, URL https://doi.org/10.1162/neco_a_01351, https://direct.mit.edu/neco/article-pdf/33/3/713/1889421/neco_a_01351.pdf

Friston KJ (2007) Statistical parametric mapping: the analysis of functional brain images. Elsevier

Friston KJ, Litvak V, Oswal A, Razi A, Stephan KE, van Wijk BC, Ziegler G, Zeidman P (2016b) Bayesian model reduction and empirical bayes for group (dcm) studies. NeuroImage 128:413–431, DOI https://doi.org/10.1016/j.neuroimage.2015.11.015, URL https://www.sciencedirect.com/science/article/pii/S105381191501037X

Friston KJ, Parr T, de Vries B (2017) The graphical brain: Belief propagation and active inference. Network Neuroscience 1(4):381–414, DOI 10.1162/NETN_a_00018, URL https://doi.org/10.1162/NETN_a_00018, https://doi.org/10.1162/NETN_a_00018

van Glabbeek RJ (1993) The linear time — branching time spectrum II. In: Best E (ed) CONCUR'93, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 66–81

Glabbeek RJv (1990) The linear time-branching time spectrum (extended abstract). In: Proceedings of the Theories of Concurrency: Unification and Extension, Springer-Verlag, Berlin, Heidelberg, CONCUR '90, p 278–297

Haarnoja T, Zhou A, Abbeel P, Levine S (2018) Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. CoRR abs/1801.01290, URL http://arxiv.org/abs/1801.01290, 1801.01290

van Hasselt H, Guez A, Silver D (2015) Deep reinforcement learning with double q-learning. CoRR abs/1509.06461, URL http://arxiv.org/abs/1509.06461, 1509.06461

Itti L, Baldi P (2009) Bayesian surprise attracts human attention. Vision Research 49(10):1295 – 1306, DOI https://doi.org/10.1016/j.visres.2008.09.007, URL http://www.sciencedirect.com/science/article/pii/S0042698908004380, visual Attention: Psychophysics, electrophysiology and neuroimaging

van de Laar T, de Vries B (2019) Simulating active inference processes by message passing. Front Robotics and AI 2019, DOI 10.3389/frobt.2019.00020, URL https://doi.org/10.3389/frobt.2019.00020

Lample G, Chaplot DS (2017) Playing FPS games with deep reinforcement learning. In: Singh SP, Markovitch S (eds) Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA, AAAI Press, pp 2140–2146, URL http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14456

Maisto D, Gregoretti F, Friston KJ, Pezzulo G (2021) Active tree search in large pomdps. CoRR abs/2103.13860, URL https://arxiv.org/abs/2103.13860, 2103.13860

Marković D, Goschke T, Kiebel SJ (2021) Meta-control of the exploration-exploitation dilemma emerges from probabilistic inference over a hierarchy of time scales. Cognitive, Affective, & Behavioral Neuroscience 21(3):509–533, DOI 10.3758/s13415-020-00837-x, URL https://doi.org/10.3758/s13415-020-00837-x

Matthey L, Higgins I, Hassabis D, Lerchner A (2017) dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/

Millidge B (2019) Combining active inference and hierarchical predictive coding: A tutorial introduction and case study. DOI 10.31234/osf.io/kf6wc, URL https://doi.org/10.31234/osf.io/kf6wc

Millidge B (2020) Deep active inference as variational policy gradients. Journal of Mathematical Psychology 96:102,348, DOI https://doi.org/10.1016/j.jmp.2020.102348, URL http://www.sciencedirect.com/science/article/pii/S0022249620300298

Millidge B, Tschantz A, Buckley CL (2021) Whence the expected free energy? Neural Comput 33(2):447–482, DOI 10.1162/neco_a_01354, URL https://doi.org/10.1162/neco_a_01354

Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller MA (2013) Playing Atari with Deep Reinforcement Learning. CoRR abs/1312.5602, URL http://arxiv.org/abs/1312.5602, 1312.5602

Parr T, Friston KJ (2018) Generalised free energy and active inference: can the future cause the past? bioRxiv DOI 10.1101/304782, URL https://www.biorxiv.org/content/early/2018/04/23/304782, https://www.biorxiv.org/content/early/2018/04/23/304782.full.pdf

Pezzato C, Corbato CH, Wisse M (2020) Active inference and behavior trees for reactive action planning and execution in robotics. CoRR abs/2011.09756, URL https://arxiv.org/abs/2011.09756, 2011.09756

Pitti A, Quoy M, Lavandier C, Boucenna S (2020) Gated spiking neural network using iterative free-energy optimization and rank-order coding for structure learning in memory sequences (inferno gate). Neural Networks 121:242–258, DOI https://doi.org/10.1016/j.neunet.2019.09.023, URL https://www.sciencedirect.com/science/article/pii/S089360801930303X

Rafetseder E, Schwitalla M, Perner J (2013) Counterfactual reasoning: From childhood to adulthood. Journal of experimental child psychology 114(3):389–404

Ren Z, Krogh B (2002) State aggregation in markov decision processes. In: Proceedings of the 41st IEEE Conference on Decision and Control, 2002., vol 4, pp 3819–3824 vol.4, DOI 10.1109/CDC.2002.1184960

Sajid N, Tigas P, Zakharov A, Fountas Z, Friston K (2021) Exploration and preference satisfaction trade-off in reward-free learning. arXiv 2106.04316

Sancaktar C, Lanillos P (2020) End-to-end pixel-based deep active inference for body perception and action. ArXiv abs/2001.05847

Sancaktar C, van Gerven MAJ, Lanillos P (2020) End-to-end pixel-based deep active inference for body perception and action. In: Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics, ICDL-EpiRob 2020, Valparaiso, Chile, October 26-30, 2020, IEEE, pp 1–8, DOI 10.1109/ICDL-EpiRob48136.2020.9278105, URL https://doi.org/10.1109/ICDL-EpiRob48136.2020.9278105

Schrittwieser J, Antonoglou I, Hubert T, Simonyan K, Sifre L, Schmitt S, Guez A, Lockhart E, Hassabis D, Graepel T, Lillicrap TP, Silver D (2019) Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. ArXiv abs/1911.08265

Schwartenbeck P, Passecker J, Hauser TU, FitzGerald THB, Kronbichler M, Friston K (2018) Computational mechanisms of curiosity and goal-directed exploration. bioRxiv DOI 10.1101/411272, URL https://www.biorxiv.org/content/early/2018/09/07/411272, https://www.biorxiv.org/content/early/2018/09/07/411272.full.pdf

Silver D, Veness J (2010) Monte-carlo planning in large pomdps. Advances in neural information processing systems 23

Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap TP, Leach M, Kavukcuoglu K, Graepel T, Hassabis D (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484–489, DOI 10.1038/nature16961, URL https://doi.org/10.1038/nature16961

Smith R, Schwartenbeck P, Parr T, Friston KJ (2020) An active inference approach to modeling structure learning: Concept learning as an example case. Frontiers in Computational Neuroscience 14:41, DOI 10.3389/fncom.2020.00041, URL https://www.frontiersin.org/article/10.3389/fncom.2020.00041

Veness J, Lanctot M, Bowling M (2011) Variance reduction in monte-carlo tree search. Advances in Neural Information Processing Systems 24

Winn J, Bishop C (2005) Variational message passing. Journal of Machine Learning Research 6:661–694

Wirkuttis N, Tani J (2021) Leading or following? dyadic robot imitative interaction using the active inference framework. IEEE Robotics and Automation Letters 6(3):6024–6031, DOI 10.1109/LRA.2021.3090015