# Multi-Modal and Multi-Factor Branching Time Active Inference ($BTAI_{3MF}$).

**Théophile Champion** TMAC3@KENT.AC.UK
*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*

**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*

**Editor: TO BE FILLED**

## 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. Recently, two versions of branching time active inference (BTAI) based on Monte-Carlo tree search have been developed to handle the exponential (space and time) complexity class that occurs when computing the prior over all possible policies up to the time horizon. However, those two versions of BTAI still suffer from an exponential complexity class w.r.t the number of observed and latent variables being modelled. In the present paper, we resolve this limitation by first allowing the modelling of several observations, each of them having its own likelihood mapping. Similarly, we allow each latent state to have its own transition mapping. The inference algorithm then exploits the factorisation of the likelihood and transition mappings to accelerate the computation of the posterior. Those two optimisations were tested on the dSprites environment in which the metadata of the dSprites dataset was used as input to the model instead of the dSprites images. On this task, $BTAI_{VMP}$ (**??**) was able to solve 96.9% of the task in 5.1 seconds, and $BTAI_{BF}$ (**?**) was able to solve 98.6% of the task in 17.5 seconds. Our new approach ($BTAI_{3MF}$) outperformed both of its predecessors by solving the task completly (100%) in only 2.559 seconds. Finally, $BTAI_{3MF}$ has been implemented in a flexible and easy to use (python) package, and we developed a graphical user interface to enable the inspection of the model's beliefs, planning process and behaviour.

**Keywords:** Branching Time Active Inference, Monte-Carlo Tree Search, Belief Propagation, Bayesian Prediction, Temporal Slice

## 1. Introduction

Active inference extends the free energy principle to generative models with actions (**???**) and can be regarded as a form of planning as inference (**?**). This framework has successfully explained a wide range of neuro-cognitive phenomena, such as habit formation (**?**), Bayesian surprise (**?**), curiosity (**?**), and dopaminergic discharges (**?**). It has also been applied to a variety of tasks, such as animal navigation (**?**), robotic control (**??**), the mountain car problem (**?**), the game DOOM (**?**) and the cart pole problem (**?**).

However, active inference suffers from an exponential (space and time) complexity class that occurs when computing the prior over all possible policies up to the time horizon. Recently, two versions of branching time active inference (BTAI) based on Monte-Carlo tree search (**?**) have been developed to handle this exponential growth. In the original formulation of the framework (**??**), inference was performed using the variational message passing (VMP)

algorithm (**??**). In a follow up paper, VMP was then replaced by a Bayesian filtering (**?**) scheme leading to a faster inference process (**?**).

In this paper, we develop an extension of Branching Time Active Inference (BTAI), to allow modelling of several modalities as well as several latent states. Indeed, even if the Bayesian filtering version of Branching Time Active Inference ($BTAI_{BF}$) is fast, its modelling capacity is limited to one observation and one hidden state. Consequently, if one wanted to model $n$ latent states $S_t^1, \dots, S_t^n$, then those $n$ latent states would have to be encoded into one latent state $X$ representing all possible configurations of the $n$ latent states $S_t^1, \dots, S_t^n$. Unfortunatly, the total number of configurations is given by:

$$\#X = \prod_{i=1}^{n} \#S_t^i \geq 2^n,$$

where $\#X$ is the number of possible values taken by $X$, and similarly $\#S_t^i$ is the number of possible values taken by $S_t^i$. The above inequality is obtained by realizing that $\#S_t^i \geq 2$, and is problematic in practice because $\#X$ is growing exponentially with the number of latent states $n$ being modelled. Also, note that in practice this exponential growth may be way worse than $2^n$. For example, if one were to model the five modalities of the dSprites environment (c.f. Section **??**), the total number of configurations would be:

$$\#S_t^y \times \#S_t^x \times \#S_t^{scale} \times \#S_t^{orientation} \times \#S_t^{scale} = 33 \times 32 \times 3 \times 40 \times 6 = 760,320 \gg 2^5 = 32.$$

A similar exponential explosion also appears when trying to model several modalities $O_t^1, \dots, O_t^m$ using a single one $Y$, i.e.

$$\#Y = \prod_{i=1}^{m} \#O_t^i \geq 2^m,$$

where $\#Y$ is the number of possible values taken by $Y$, and similarly $\#O_t^i$ is the number of possible values taken by $O_t^i$. Note, throughout this paper, we will use the term *states* to refer to the latent states of the model at a specify time step, e.g., $S_t^1, \dots, S_t^n$ for time step $t$. Additionally, we will use the terms *states configurations* or *values* to refer to particular values taken by the latent variables.

The present paper aims to remove those two exponential growths, by allowing the modelling of several observations and latent states, while providing an easy to use framework based on a high-level notational language, which allows the user to create models by simply declaring the variables it contains, and the dependencies between those variables. Then, the framework performs the inference process automatically. Appendix A shows an example of how to implement a custom $BTAI_{3MF}$ agent using our framework. In section **??**, we describe the theory underlying our approach. Importantly, $BTAI_{3MF}$ takes advantage of the generative model struture to perform inference efficiently using a mixture of belief propagation (**???**) and forward predictions as will be explained in Section **??**. Next, in Section **??**, we provide the definition of the expected free energy in the context of our new approach, and in Section **??**, we describe the planning algorithm used to expand the generative model dynamically. Then, in Section **??**, we compare $BTAI_{3MF}$ to $BTAI_{VMP}$ and $BTAI_{BF}$, and demonstrate empirically that $BTAI_{3MF}$ outperformed both $BTAI_{VMP}$ and $BTAI_{BF}$ on the dSprites environment, which requires the modelling of many latent states and modalities. Finally, Section **??** concludes this paper by summarizing our approach and results.

## 2. Theory of $BTAI_{3MF}$

In this section, we introduce the mathematical foundation of $BTAI_{3MF}$. To simplify the graphical representation of our generative model, we first introduce a notion of "temporal slice". Then, we build on this idea to describe the generative model of $BTAI_{3MF}$. Next, we explain how belief updates are performed using a mixture of belief propagation and forward predictions. Afterwards, we provide the definition of the expected free energy for this new generative model. Finally, we describe the planning algorithm used to dynamically expand the generative model, and the action selection process.

### 2.1 Temporal slice

A temporal slice $TS_J = \{O_J^1, \ldots, O_J^{\#O}, S_J^1, \ldots, S_J^{\#S}\}$ is a set of random variables indexed by a sequence of actions $J$. Each random variable of the temporal slice represents either an observation $O_J^o$ or a latent state $S_J^s$. The index of the temporal slice correponds to the sequence of actions that lead to this temporal slice. By definition, if $J$ is an empty sequence, i.e., $J = \emptyset$, then $TS_J$ is the temporal slice of the present time step $t$, also denoted $TS_t$. Within a temporal slice $TS_J$, an observation $O_J^o$ depends on a number of latent states $\rho_J^o \subseteq \{S_J^s \mid s = 1, \ldots, \#S\}$, such that $P(O_J^o|\rho_J^o)$ is a factor in the generative model. Given an action $\boldsymbol{a}$ and a sequence of actions $J$, we let $I = J{::}\boldsymbol{a}$ be the sequence of actions obtained by appending the action $\boldsymbol{a}$ at the end of the sequence of actions $J$. If $I = J{::}\boldsymbol{a}$, then the temporal slice $TS_J$ can be the parent of $TS_I$. This means that a latent state $S_I^s$ in $TS_I$ can depend on the latent states $\rho_I^s \subseteq \{S_J^s \mid s = 1, \ldots, \#S\}$ in $TS_J$, such that $P(S_I^s|\rho_I^s)$ is a factor in the generative model. The concept of temporal slice is illustrated in Figure **??**, and Figure **??** depicts a more compact representation of the content of Figure **??**.

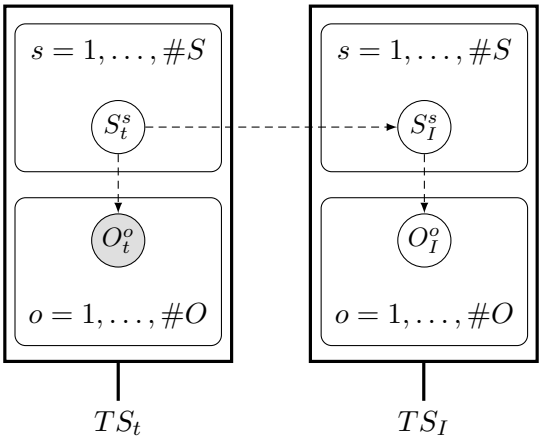



Figure 1: This figure illustrates two temporal slices $TS_t$ and $TS_I$, which are depicted by rectangles with thick border. Within each temporal slice, plate notation is used to generate $\#S$ latent states and $\#O$ observations. The dashed lines that connect two random variables from two different plates are new to this paper, and represent an arbitrary connectivity between the two sets of random variables generated by the plates. For example, the dashed line from $S_t^s$ to $O_t^o$, means that for each observation $O_t^o$, the parents of $O_t^o$ denoted $\rho_t^o$ is a subset of $\{S_t^s \mid s = 1, \ldots, \#S\}$, i.e., the generative model contains the factor $P(O_t^o|\rho_t^o)$ where $\rho_t^o \subseteq \{S_t^s \mid s = 1, \ldots, \#S\}$.

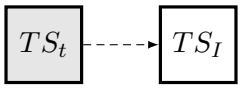



Figure 2: This figure illustrates the two temporal slices $TS_t$ and $TS_I$ from Figure **??** in a more compact fashion. Since $O_t^o$ is an observed variable for all $o \in \{1, \ldots, \#O\}$, the square representing $TS_t$ has a gray background. In contrast, the square representing $TS_I$ has a white background because $O_I^o$ is a latent variable for all $o \in \{1, \ldots, \#O\}$.

### 2.2 Generative model

In this section, we build upon the notion of temporal slice to describe the full generative model. Intuitively, the probability of the entire generative model is the product of the probability of each temporal slice within the model. This includes the current temporal slice $TS_t$ and the future temporal slices $TS_I$ for all $I \in \mathbb{I}$, where $\mathbb{I}$ is the set of all multi-indices expanded during the tree search (c.f., Section **??**). Within each temporal slice, there are $\#O$ observations and $\#S$ latent states. Each observation depends on a subset of the latent states. Moreover, each latent state depends on a subset of the latent states of the parent temporal slice. Note, the current temporal slice $TS_t$ does not have any parents, therefore its latent state does not depend on anything. In other words, the model makes the Markov assumption, i.e., each state only depends on the states at the previous time step. More formally, the

generative model is defined as:

$$P(O_t, S_t, O_{\mathbb{I}}, S_{\mathbb{I}}) = P(TS_t) \prod_{I \in \mathbb{I}} P(TS_I)$$
$$= \underbrace{\prod_{o=1}^{\#O} P(O_t^o|\rho_t^o) \prod_{s=1}^{\#S} P(S_t^s)}_{\text{current temporal slice } TS_t} \prod_{I \in \mathbb{I}} \Bigg[ \underbrace{\prod_{o=1}^{\#O} P(O_I^o|\rho_I^o) \prod_{s=1}^{\#S} P(S_I^s|\rho_I^s)}_{\text{future temporal slice } TS_I} \Bigg]$$

where $t$ is the current time step, $\rho_\tau^x$ is the set of parents of $X_\tau^x$, $O_t = \{O_t^o \mid o = 1, \ldots, \#O\}$ is the set of all observations at time $t$, $O_I = \{O_I^o \mid o = 1, \ldots, \#O\}$ is the set of all future observations that would be observed after performing the sequence of actions $I$, $O_{\mathbb{I}} = \cup_{I \in \mathbb{I}} O_I$ is the set of all future observations contained in the temporal slices expanded during the tree search (c.f., Section **??**), $S_t = \{S_t^s \mid s = 1, \ldots, \#S\}$ is the set of all latent states at time $t$, $S_I = \{S_I^s \mid s = 1, \ldots, \#S\}$ is the set of random variables describing the future latent states after performing the sequence of actions $I$, $S_{\mathbb{I}} = \cup_{I \in \mathbb{I}} S_I$ is the set of latent variables representing all future states contained in the temporal slices expanded during the tree search (c.f., Section **??**). Importantly, the above generative model has to satisfy:

- $\forall I \in \mathbb{I}, \forall o \in \{1, \ldots, \#O\}, \rho_I^o \subseteq S_I$;
- $\forall I{::}\boldsymbol{a} \in \mathbb{I}, \forall s \in \{1, \ldots, \#S\}, \rho_{I::\boldsymbol{a}}^s \subseteq S_I$, also, if $I = \emptyset$ then by definition $S_I \triangleq S_t$.

Additionally, we define the factors of the generative model as:

$$P(O_t^o|\rho_t^o) = \text{Cat}(\boldsymbol{A}^o), \qquad P(S_t^s) = \text{Cat}(\boldsymbol{D}_t^s),$$
$$P(O_I^o|\rho_I^o) = \text{Cat}(\boldsymbol{A}^o), \qquad P(S_I^s|\rho_I^s) = \text{Cat}(\boldsymbol{B}_I^s),$$

where $\boldsymbol{A}^o$ is the tensor modelling the likelihood mapping of the $o$-th observation, $\boldsymbol{D}_t^s$ is the vector modelling the prior over the $s$-th latent state at time $t$ (see below for details), $\boldsymbol{B}^s$ is the tensor modelling the transition mapping of the $s$-th latent state under each possible action, $\boldsymbol{B}_I^s$ is the tensor modelling the transition mapping of the $s$-th latent state under the last action $I_{\text{last}}$ of the sequence $I$, i.e., $\boldsymbol{B}_I^s = \boldsymbol{B}^s(\bullet, \ldots, \bullet, I_{\text{last}})$. Also, note that at the beginning of a trial, i.e., when $t = 0$, $\boldsymbol{D}_t^s$ is a vector that encodes the modeller's understanding of the task. Afterwards, when $t > 0$, $\boldsymbol{D}_t^s$ is a vector containing the parameters of the posterior over hidden states according to the observations made and actions taken so far, i.e., $P(S_t^s) \triangleq P(S_t^s|O_{0:t-1}, A_{0:t-1}) = \text{Cat}(\boldsymbol{D}_t^s)$ for all $s \in \{1, \ldots, \#S\}$. Finally, Figure **??** illustrates the full generative model using the notion of temporal slices.

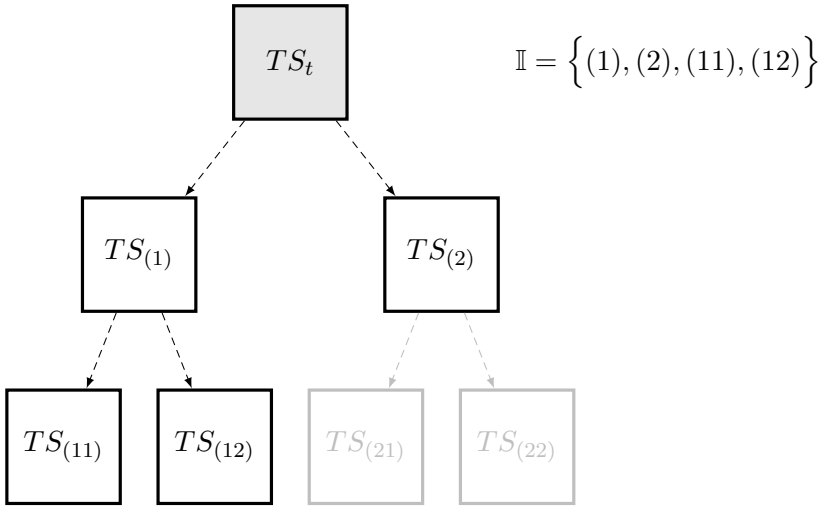



Figure 3: This figure illustrates the full generative model of $BTAI_{3MF}$. The temporal slices depited in light gray correspond to temporal slices that have not yet been explored by the planning algorithm, c.f., Section **??**. The numbers between parentheses correspond to the sequence of actions performed to reach the temporal slice.

### 2.3 Belief updates: the inference and prediction (IP) algorithm

The IP algorithm is composed of two steps, i.e., the inference step (or I-step) and the prediction step (or P-step). The goal of the I-step is to compute the posterior beliefs over all the latent variables at time $t$. In other words, the goal of the I-step is to compute: $P(S_t^s|O_t), \forall s \in \{1, \dots, \#S\}$. The P-step takes as inputs the posterior beliefs over all the latent variables corresponding to the states of the system after performing a sequence of actions $I$, and an action $\boldsymbol{a}$ to be performed next. The goal of the P-step is to compute the posterior beliefs over all the latent variables corresponding to the future states and observations after performing the sequence of actions $I{::}\boldsymbol{a}$, where $I{::}\boldsymbol{a}$ is the sequence of actions obtained by adding the action $\boldsymbol{a}$ at the end of the sequence of actions $I$. In other words, given $P(S_I^s|O_t), \forall s \in \{1, \dots, \#S\}$ and an action $\boldsymbol{a}$, the goal of the P-step is to compute: $P(S_{I{::}\boldsymbol{a}}^s|O_t), \forall s \in \{1, \dots, \#S\}$ and $P(O_{I{::}\boldsymbol{a}}^o|O_t), \forall o \in \{1, \dots, \#O\}$. Note that by definition, we let $P(S_I^m|O_t) \triangleq P(S_t^m|O_t)$ if $I = \emptyset$. To derive the inference and prediction steps, the following sections make use of the sum-rule, product-rule, and d-separation criterion (c.f., Appendix C for details about those properties).

#### 2.3.1 Inference step

As just stated, the goal of the I-step is to compute $P(S_t^m|O_t), \forall m \in \{1, \dots, \#S\}$. First, we re-write the posterior computation to fit the kind of problem that belief propagation — also known as the sum-product algorithm — can solve:

$$
\begin{aligned}
P(S_t^m|O_t) &\propto P(S_t^m, O_t) && \text{(Bayes theorem)} \\
&= \sum_{\sim S_t^m} P(S_t, O_t) && \text{(sum rule)} \\
&= \sum_{\sim S_t^m} \prod_{o=1}^{\#O} P(O_t^o|\rho_t^o) \prod_{s=1}^{\#S} P(S_t^s) && \text{(product rule \& d-separation)}
\end{aligned}
$$

where $S_t = \{S_t^s \mid s = 1, \dots, \#S\}$ is the set of all latent states at time $t$, $\sim S_t^m = S_t \setminus S_t^m$ is the set of all latent states at time $t$ except $S_t^m$, and the summation is over all possible configurations of $\sim S_t^m$, i.e., we are marginalizing out all states, apart from one; thus $P(S_t, O_t)$ has $\#S + \#O$ dimensions, while $P(S_t^m, O_t)$ has $1 + \#O$ dimensions. Since $\rho_t^o \subseteq S_t$, the expression inside the summation is a function $g(S_t)$ that factorizes as follows:

$$
\begin{aligned}
g(S_t) &= \prod_{o=1}^{\#O} P(O_t^o|\rho_t^o) \prod_{s=1}^{\#S} P(S_t^s) \\
&\triangleq \prod_{i=1}^{N} f_i(X_i),
\end{aligned}
$$

where $X_i \subseteq S_t$ for all $i \in \{1, \dots, \#O + \#S\}$, the number of factors is $N = \#O + \#S$, and:

$$
f_i(X_i) \triangleq \begin{cases} P(O_t^i|\rho_t^i) & \text{if } i \in \{1, \dots, \#O\} \\ P(S_t^{i-\#O}) & \text{if } i \in \{\#O + 1, \dots, \#O + \#S\} \end{cases}.
$$

Note that, because $O_t^o$ (denoted $O_t^i$ here) are known constants, we do not specify that $g(S_t)$ depends on $O_t^o$. To conclude, by substituting the definition of $g(S_t)$ into the formula of the posterior $P(S_t^m|O_t)$ presented above, we get:

$$
P(S_t^m|O_t) \propto \sum_{\sim S_t^m} g(S_t),
$$

which means that the posterior $P(S_t^m|O_t)$ can be computed by first marginalizing $g(S_t)$ w.r.t. $S_t^m$, i.e.,

$$g(S_t^m) = \sum_{\sim S_t^m} g(S_t),$$

and then normalizing:

$$P(S_t^m|O_t) = \frac{g(S_t^m)}{\sum_{S_t^m} g(S_t^m)}.$$

The marginalization of $g(S_t)$ can be performed efficiently using belief propagation (**?**), which can be understood as a message passing algorithm on a factor graph. The message from a node $x$ to a factor $f$ is given by:

$$m_{x\to f}(x) = \prod_{h\in n(x)\setminus\{f\}} m_{h\to x}(x),$$

where $n(x)$ are the neighbours of $x$ in the factor graph. Note, in a factor graph the neighbours of a random variable are factors. Moreover, the message from a factor $f$ to a node $x$ is given by:

$$m_{f\to x}(x) = \sum_{Y} \left( f(X) \prod_{y\in Y} m_{y\to f}(y) \right),$$

where $X = n(f)$ are the neighbours of $f$ in the factor graph, $Y = X \setminus \{x\}$ are all the neighbours of $f$ except $x$, and the summation is over all possible configurations of the variables in $Y$. Note, in a factor graph the neighbours of a factor are random variables. Once all the messages have been computed, the marginalization of $g(S_t)$ w.r.t. $S_t^m$ is given by the product of all the incoming messages of the node $S_t^m$, i.e.,

$$g(S_t^m) = \prod_{f\in n(S_t^m)} m_{f\to S_t^m}(S_t^m).$$

### 2.3.2 Prediction step

The P-step is analogous to the prediction step of Bayesian filtering (**?**). Given $P(S_I^s|O_t)$ for each $s \in \{1,\dots,\#S\}$ and an action $\boldsymbol{a}$, the goal of the P-step is to compute $P(S_{I::\boldsymbol{a}}^s|O_t)$ for each latent state $s \in \{1,\dots,\#S\}$ and $P(O_{I::\boldsymbol{a}}^o|O_t)$ for each future observation $o \in \{1,\dots,\#O\}$. For the sake of brevity, we let $J \triangleq I::\boldsymbol{a}$. Let's start with the computation of $P(S_{I::\boldsymbol{a}}^s|O_t)$:

$$\begin{aligned}
P(S_{I::\boldsymbol{a}}^s|O_t) \triangleq P(S_J^s|O_t) &= \sum_{\rho_J^s} P(S_J^s, \rho_J^s|O_t) && \text{(sum rule)}\\
&= \sum_{\rho_J^s} P(S_J^s|\rho_J^s, O_t) P(\rho_J^s|O_t) && \text{(product rule)}\\
&= \sum_{\rho_J^s} P(S_J^s|\rho_J^s) P(\rho_J^s|O_t) && \text{(d-separation)}\\
&\approx \sum_{\rho_J^s} P(S_J^s|\rho_J^s) \prod_{i=1}^{\#\rho_J^s} P(\rho_{J,i}^s|O_t) && \text{(mean-field approximation)}
\end{aligned}$$

where $\#\rho_J^s$ is the number of parents of $S_J^s$, and $\rho_{J,i}^s$ is the $i$-th parent of $S_J^s$. Importantly, $P(S_J^s|\rho_J^s)$ is known from the definition of the generative model. Moreover, since $\rho_{J,i}^s \in S_I$, then $P(\rho_{J,i}^s|O_t) = P(S_I^m|O_t)$ for some $m \in \{1,\dots,\#S\}$.

Thus, $P(\rho^s_{J,i}|O_t)$ is given as input to the P-step, i.e., $P(\rho^s_{J,i}|O_t)$ is a known distribution. Similarly, the computation of $P(O^o_{I::\boldsymbol{a}}|O_t)$ proceeds as follows:

$$
\begin{aligned}
P(O^o_{I::\boldsymbol{a}}|O_t) \triangleq P(O^o_J|O_t) &= \sum_{\rho^o_J} P(O^o_J, \rho^o_J|O_t) && \text{(sum rule)} \\
&= \sum_{\rho^o_J} P(O^o_J|\rho^o_J, O_t) P(\rho^o_J|O_t) && \text{(product rule)} \\
&= \sum_{\rho^o_J} P(O^o_J|\rho^o_J) P(\rho^o_J|O_t) && \text{(d-separation)} \\
&\approx \sum_{\rho^o_J} P(O^o_J|\rho^o_J) \prod_{i=1}^{\#\rho^o_J} P(\rho^o_{J,i}|O_t) && \text{(mean-field approximation)}
\end{aligned}
$$

where $\#\rho^o_J$ is the number of parents of $O^o_J$, and $\rho^o_{J,i}$ is the $i$-th parent of $O^o_J$. Importantly, $P(O^o_J|\rho^o_J)$ is known from the definition of the generative model. Moreover, since $\rho^o_{J,i} \in S_J$, then $P(\rho^o_{J,i}|O_t) = P(S^s_J|O_t)$ for some $s \in \{1, \ldots, \#S\}$. Thus, $P(\rho^o_{J,i}|O_t)$ has already been computed during the first stage of the P-step and is a known distribution, c.f., derivation of $P(S^s_{I::\boldsymbol{a}}|O_t) \triangleq P(S^s_J|O_t)$.

### 2.4 Expected Free Energy

In this section, we discuss the definition of the expected free energy, which quantifies the cost of pursuing a particular sequence of actions and will be useful for planning, cf. Section **??**. The expected free energy (see below) is composed of the risk and ambiguity terms. The risk terms quantify how much the posterior beliefs over future observations (computed by the P-step) diverge from the prior preferences of the agent. On the other hand, the ambiguity terms correspond to the expected uncertainty of the likelihood mapping, where the expectation is with respect to the posterior beliefs over states computed by the P-step.

First, we partition the set of observations $O_I = \{O^o_I \mid o = 1, \ldots, \#O\}$ into disjoint subsets $X^I_i$, i.e., $O_I = X^I_1 \cup \ldots \cup X^I_N$ and $X^I_i \cap X^I_j = \emptyset$ if $i \neq j$. Then, we define the prior preferences over the $i$-th subset of observations as: $V(X^I_i) = \text{Cat}(\boldsymbol{C}^i)$. This formulation allows us to define prior preferences over subsets of random variables, and will be useful in Section **??**, where the agent needs to possess preferences that depend upon both the shape and $(X, Y)$ position of the object. Finally, the expected free energy, which needs to be minimised, is given by:

$$
\boldsymbol{G}_I \triangleq \sum_{i=1}^{N} \left( \underbrace{D_{\text{KL}}[P(X^I_i|O_t)||V(X^I_i)]}_{\text{risk of } i\text{-th set of observations}} \right) + \sum_{o=1}^{\#O} \left( \underbrace{\mathbb{E}_{P(\rho^o_I|O_t)}[\text{H}[P(O^o_I|\rho^o_I)]]}_{\text{ambiguity of } o\text{-th observation}} \right), \tag{1}
$$

where $P(X^I_i|O_t)$ and $P(\rho^o_I|O_t)$ are the posteriors over the $i$-th subset of observations and the parent of $O^o_I$, respectively, and $P(O^o_I|\rho^o_I)$ is known from the generative model. Assuming a mean-field approximation, those posteriors are given by:

$$
\begin{aligned}
P(\rho^o_I|O_t) &\approx \prod_{i=1}^{\#\rho^o_I} P(\rho^o_{I,i}|O_t) \\
P(X^I_i|O_t) &\approx \prod_{O^o_I \in X_i} P(O^o_I|O_t)
\end{aligned}
$$

where $P(O^o_I|O_t)$ and $P(\rho^o_{I,i}|O_t)$ are the posteriors over $O^o_I$ and the $i$-th parent of $O^o_I$, respectively. Note, both $P(O^o_I|O_t)$ and $P(\rho^o_{I,i}|O_t)$ were computed during the P-step. The definition of the expected free energy given by (**??**) may not

be very intuitive. Fortunatly, the special case where each subset contains a single observation, i.e., $X_o^I = O_I^o$, leads to the following equation:

$$\boldsymbol{G}_I \triangleq \sum_{o=1}^{\#O} \Bigg( \underbrace{D_{\mathrm{KL}}[P(O_I^o|O_t)||V(O_I^o)]}_{\text{risk of } o\text{-th observation}} + \underbrace{\mathbb{E}_{P(\rho_I^o|O_t)}[\mathrm{H}[P(O_I^o|\rho_I^o)]]}_{\text{ambiguity of } o\text{-th observation}} \Bigg),$$

which is the summation over all observations $O_I^o$ of the expected free energy of $O_I^o$, i.e., the risk of $O_I^o$ plus the ambiguity of $O_I^o$. Finally, our framework allows to specify prior preferences over only a subset of variables in $O_I$. For example, if a task contains four variables, i.e., $O_I^x$, $O_I^y$, $O_I^{shape}$ and $O_I^{scale}$, but it only makes sense to have preferences over three of them, i.e., $O_I^x$, $O_I^y$ and $O_I^{shape}$, then the prior preference over the fourth variable is set to the posterior over this random variable, i.e., $V(O_I^{shape}) \triangleq P(O_I^{shape}|O_t)$. In other words, not having prior preferences over a random variable is viewed by our framework as liking whatever we predict will happen. Effectively, this renders the risk term associated with such variable equal to zero, i.e.,

$$D_{\mathrm{KL}}[P(O_I^{shape}|O_t)||V(O_I^{shape})] = D_{\mathrm{KL}}[P(O_I^{shape}|O_t)||P(O_I^{shape}|O_t))] = 0.$$

### 2.5 Planning: the MCTS algorithm

In this section, we describe the planning algorithm used by $BTAI_{3MF}$. At the beginning of a trial when $t = 0$, the agent is provided with the initial observations $O_0$. The I-step is performed and returns the posterior over all latent states, i.e., $P(S_0^s|O_0)$ for all $s \in \{1, \ldots, \#S\}$, according to the prior over the initial hidden states provided by the modeller, i.e., $P(S_0^s)$ for all $s \in \{1, \ldots, \#S\}$, and the available observations $O_0$.

Then, we use the UCT criterion to determine which node in the tree should be expanded. Let the tree's root $TS_t$ be called the current node. If the current node has no children, then it is selected for expansion. Alternatively, the child with the highest UCT criterion becomes the new current node and the process is iterated until we reach a leaf node (i.e. a node from which no action has previously been selected). The UCT criterion (**?**) for the $j$-th child of the current node is given by:

$$UCT_j = -\bar{\boldsymbol{G}}_j + C_{explore}\sqrt{\frac{\ln n}{n_j}}, \tag{2}$$

where $\bar{\boldsymbol{G}}_j$ is the average expected free energy calculated with respected to the actions selected from the $j$-th child, $C_{explore}$ is the exploration constant that modulates the amount of exploration at the tree level, $n$ is the number of times the current node has been visited, and $n_j$ is the number of times the $j$-th child has been visited.

Let $S_I$ be the (leaf) node selected by the above selection procedure. We then expand all the children of $S_I$, i.e., all the states of the form $S_{I::\boldsymbol{a}}$, where $\boldsymbol{a} \in \{1, ..., \#A\}$ is an arbitrary action, $\#A$ is the number of available actions, and $I::\boldsymbol{a}$ is the multi-index obtained by appending the action $\boldsymbol{a}$ at the end of the sequence defined by $I$. Next, we perform the P-step for each action $\boldsymbol{a}$, and obtain $P(S_{I::\boldsymbol{a}}^s|O_t)$ for each latent state $s \in \{1, \ldots, \#S\}$ and $P(O_{I::\boldsymbol{a}}^o|O_t)$ for each future observation $o \in \{1, \ldots, \#O\}$.

Then, we need to estimate the cost of (virtually) taking each possible action. The cost in this paper is taken to be the expected free energy given by (**??**). Next, we assume that the agent will always perform the action with the lowest cost, and back-propagate the cost of the best (virtual) action toward the root of the tree. Formally, we write the update as follows:

$$\forall K \in \mathbb{A}_I \cup \{I\}, \quad \boldsymbol{G}_K \leftarrow \boldsymbol{G}_K + \min_{\boldsymbol{a} \in \{1,\ldots,\#A\}} \boldsymbol{G}_{I::\boldsymbol{a}}, \tag{3}$$

where $I$ is the multi-index of the node that was selected for (virtual) expansion, and $\mathbb{A}_I$ is the set of all multi-indices corresponding to ancestors of $TS_I$. During the back propagation, we also update the number of visits as follows:

$$\forall K \in \mathbb{A}_I \cup \{I\}, \quad n_K \leftarrow n_K + 1. \tag{4}$$

If we let $\boldsymbol{G}_K^{aggr}$ be the aggregated cost of an arbitrary node $S_K$ obtained by applying Equation **??** after each expansion, then we are now able to express $\bar{\boldsymbol{G}}_K$ formally as:

$$\bar{\boldsymbol{G}}_K = \frac{\boldsymbol{G}_K^{aggr}}{n_K}.$$

The planning procedure described above ends when the maximum number of planning iterations is reached.

### 2.6 Action selection

After performing planning, the agent needs to choose the action to perform in the environment. As discussed in Section 3.1 of (**?**), many possible mechanisms can be used to select the action to perform in the environment. $BTAI_{3MF}$ performs the action corresponding to the root child with the highest number of visits. Formally, this is expressed as:

$$\boldsymbol{a}^* = \underset{\boldsymbol{a} \in \{1,\dots,\#A\}}{\arg\max}\ n_{(\boldsymbol{a})}, \tag{5}$$

where $\boldsymbol{a}^*$ is the action performed in the environment, and $n_{(\boldsymbol{a})}$ is the number of visits of the root child corresponding to action $\boldsymbol{a}$.

### 2.7 Closing the action-perception cycle

After performing an action $\boldsymbol{a}^*$ in the environment, the agent receives a new observation $O_{t+1}$, and needs to use this observation to compute the posterior over the latent states at time $t+1$, i.e., $P(S_{t+1}^s|O_{t+1})$ for all $s \in \{1,\dots,\#S\}$. This can be achieved by performing the I-step, but requires the agent to have prior beliefs over the latent states at time $t+1$, i.e., $P(S_{t+1}^s)$ for all $s \in \{1,\dots,\#S\}$, in addition to the new observation $O_{t+1}$ obtained from the environment. In this paper, we define those prior beliefs as:

$$P(S_{t+1}^s) = P(S_I^s|O_t), \text{ for all } s \in \{1,\dots,\#S\},$$

where $I = (\boldsymbol{a}^*)$ is a sequence of actions containing the action $\boldsymbol{a}^*$ performed in the environment, $P(S_I^s|O_t)$ is the predictive posterior computed by the P-step when assuming that action $\boldsymbol{a}^*$ is performed. In other words, the predictive posterior $P(S_I^s|O_t)$ computed by the P-step at time $t$, is used as an empirical prior $P(S_{t+1}^s)$ at time $t+1$. This empirical prior $P(S_{t+1}^s)$ along with the new observation $O_{t+1}$ can then be used to compute the posterior $P(S_{t+1}^s|O_{t+1})$ for all

$s \in \{1, \dots, \#S\}$. This posterior will be used to perform planning in the next action-perception cycle. Algorithm **??** concludes this section by summarizing our approach.

**Algorithm 1:** $BTAI_{3MF}$: action-perception cycles (with relevant equations indicated in round brackets).

**Input:** *env* the environment,
$O_0 = \{O_0^o \mid o = 1, \dots \#O\}$ the initial observations,
$\boldsymbol{A} = \{\boldsymbol{A}^o \mid o = 1, \dots \#O\}$ the likelihood mapping of each observation,
$\boldsymbol{B} = \{\boldsymbol{B}^s \mid s = 1, \dots, \#S\}$ the transition mapping for each hidden state,
$\boldsymbol{C} = \{\boldsymbol{C}^i \mid i = 1, \dots N\}$ the prior preferences of each subset of observations,
$\boldsymbol{D}_0 = \{\boldsymbol{D}_0^s \mid s = 1, \dots \#S\}$ the prior over each initial state,
$N$ the number of planning iterations,
$M$ the number of action-perception cycles.

$P(S_0^s|O_0) \leftarrow$ I-step($O_0$, $\boldsymbol{A}$, $\boldsymbol{D}_0$) `// I-step from Section ??`
*root* $\leftarrow$ CreateTreeNode(
  beliefs = $P(S_0^s|O_0)$, action = -1, cost = 0, visits = 1
) `// Create the root node for the MCTS, where -1 is a dummy value`
**repeat** $M$ **times**
  **repeat** $N$ **times**
    *node* $\leftarrow$ SelectNode(*root*) `// Using (??) recursively`
    *eNodes* $\leftarrow$ ExpandChildren(*node*, $\boldsymbol{B}$) `// P-step from Section ?? for each action`
    Evaluate(*eNodes*, $\boldsymbol{A}$, $\boldsymbol{C}$) `// Compute (??) for each expanded node`
    Backpropagate(*eNodes*) `// Using (??) and (??)`
  **end**
  $\boldsymbol{a}^* \leftarrow$ SelectAction(*root*) `// Using (??)`
  $O_{t+1} \leftarrow$ *env*.Execute($\boldsymbol{a}^*$)
  *child* $\leftarrow$ *root.children*[$\boldsymbol{a}^*$] `// Get root child corresponding to` $\boldsymbol{a}^*$
  $P(S_{t+1}^s) \leftarrow$ *child.beliefs* `// Get the empirical prior` $P(S_{t+1}^s) = \texttt{Cat}(\boldsymbol{D}_{t+1}^s)$
  $P(S_{t+1}^s|O_{t+1}) \leftarrow$ I-step($O_{t+1}$, $\boldsymbol{A}$, $\boldsymbol{D}_{t+1}$) `// I-step from Section ??`
  *root* $\leftarrow$ CreateTreeNode(
    beliefs = $P(S_{t+1}^s|O_{t+1})$, action = $\boldsymbol{a}^*$, cost = 0, visits = 1
  ) `// Create the root node of the next action-perception cycle`
**end**

## 3. Results

In this section, we compare our new approach to BTAI with variational message passing ($BTAI_{VMP}$) and BTAI with Bayesian filtering ($BTAI_{BF}$). Section **??** presents the simplified version of the dSprites environment on which the agents are compared. Section **??** describes how the task is modelled by the $BTAI_{VMP}$ agent and reports its performance, finally, Sections **??** and **??** do the same for the $BTAI_{BF}$ and $BTAI_{3MF}$ agents. For the reader interested in implementing a custom $BTAI_{3MF}$ agent, Appendix A provides a tutorial of how to create such an agent using our framework, and Appendix B desbribes a graphical user interface (GUI) that can be used to inspect the model. This GUI displays the structure of the generative model and prior preferences, the posterior beliefs of each latent variable, the messages sent throughout the factor graph to perform inference, the information related to the MCTS algorithm, and the expected free energy (EFE) of each node in the future. It also shows how the EFE decomposes into the risk and ambiguity terms.

### 3.1 dSprites Environment

The dSprites environment is based on the dSprites dataset (**?**) initially designed for analysing the latent representation learned by variational auto-encoders (**?**). The dSprites dataset is composed of images of squares, ellipses and hearts. Each image contains one shape (square, ellipse or heart) with its own scale, 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 **??**.

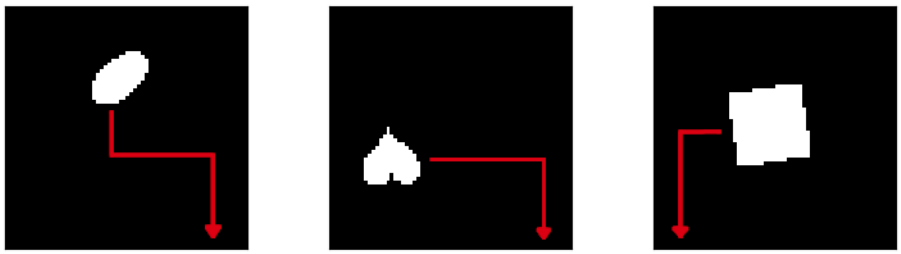
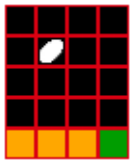


Figure 4: This figure illustrates the dSprites environment, in which the agent must move all squares towards the bottom-left corner of the image and all ellipses and hearts towards the bottom-right corner of the image. The red arrows show the behaviour expected from the agent.

Since BTAI is a tabular model whose likelihood and transition 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), the scale and orientation of the shape, 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 $16 \times 16$ images, and thus, the positions at coordinate $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$ are indistinguishable. Figure **??** illustrates the coarse grained representation with a granularity of eight and the corresponding indices observed by the $BTAI_{VMP}$ and $BTAI_{BF}$ agents. Note that this modification of the observation space can be seen as a form of state aggregation (**?**). Finally, as shown in Figure **??**, the prior preferences of the agent are specified over an absorbing row below the dSprites image. This absorbing 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.



| | | | |
|---|---|---|---|
| 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 5: 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 $8 \times 8$ pixels. Any positions in those $8 \times 8$ squares are indistinguishable from the perspective of the agent. Also, the bottom most row is an absorbing 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 $BTAI_{VMP}$ and $BTAI_{BF}$ agents 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 absorbing row or if it does so at the antipode of the appropriate corner. As the agent enters the absorbing 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 absorbing row or doing so at the antipode of the appropriate corner. In contrast, a score of one corresponds to an agent always entering the absorbing row through the appropriate corner.

### 3.2 $BTAI_{VMP}$ modeling approach and results

In this section, we evaluate $BTAI_{VMP}$ (**??**) on the dSprites environment. As shown in Figure **??**, $BTAI_{VMP}$ observes one index for each possible configuration of shape, and $(X, Y)$ positions. Importantly, this version of BTAI suffers from the exponential growth described in the introduction, and thus does not models the scale and orientation modalities. Also, to make the inference and planning process tractable, the granularity of the coarse-grained representation was set to four or eight. Table **??** provides the value of each hyper-parameter used by $BTAI_{VMP}$ in this section. Note, the hyper-parameters values are the same for all BTAI models presented in this paper. Only the number of action perception cycles, and the number of planning iterations may vary from one experience to the next.

| 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 1: The value of each hyper-parameter used by $BTAI_{VMP}$ in this section. `NB_SIMULATIONS` is the number of simulations run during the experiment. `NB_ACTION_PERCEPTION_CYCLES` is the maximum number of actions executed in each simulation, after which the simulation is terminated. `NB_PLANNING_STEPS` is the number of planning iterations performed by the agent. `EXPLORATION_CONSTANT` is the exploration constant of the UCT criterion. `PRECISION_PRIOR_PREFERENCES` is the precision of the prior preferences. `PRECISION_ACTION_SELECTION` is the precision of the distribution used for action selection. `EVALUATION_TYPE` is the type of cost used to evaluate the node during the tree search. Those hyper-parameters can be used to re-run the experiments using the code of the following GitHub repository: `https://github.com/ChampiB/Experiments_AI_TS`.

Briefly, the agent is able to solve 88.5% of the task when using a granularity of eight, c.f. Table **??**. To understand why $BTAI_{VMP}$ 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 **??**. 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 receive the maximum reward because its true $X$ position is 24 instead of the optimal $X$ position of 31.

| Planning iterations | P(solved) | Time (sec) |
|---|---|---|
| 10 | 0.813 | $0.859 \pm 0.868$ |
| 25 | 0.846 | $0.862 \pm 0.958$ |
| 50 | 0.885 | $1.286 \pm 1.261$ |

Table 2: The percentage of the dSprites environment solved by the $BTAI_{VMP}$ agent when using a granularity of eight, c.f. Figure **??**. The last column reports the average execution time required for one simulation and the associated standard deviation.

As shown in Table **??**, 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 **??**. 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 a result, 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 | 0.859 | $3.957 \pm 4.027$ |
| 25 | 0.933 | $3.711 \pm 4.625$ |
| 50 | 0.969 | $5.107 \pm 5.337$ |

Table 3: The percentage of the dSprites environment solved by the $BTAI_{VMP}$ 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 for one simulation and the associated standard deviation.

### 3.3 $BTAI_{BF}$ modeling approach and results

In this section, we evaluate $BTAI_{BF}$ (**?**) on the dSprites environment. As shown in Figure **??**, $BTAI_{BF}$ observes one index for each possible configuration of shape, and $(X, Y)$ positions. Also, to make the inference and planning process tractable, the granularity of the coarse-grained representation was set to two, four or eight. Table **??** provides the value of each hyper-parameter used by $BTAI_{BF}$ in this section. Note, the hyper-parameters values are the same for all BTAI models presented in this paper. Only the number of action perception cycles, and the number of planning iterations may vary from one experience to the next.

| Name | Value |
|---|---|
| `NB_SIMULATIONS` | 100 |
| `NB_ACTION_PERCEPTION_CYCLES` | 20 |
| `NB_PLANNING_STEPS` | 50 |
| `EXPLORATION_CONSTANT` | 2.4 |
| `PRECISION_PRIOR_PREFERENCES` | 1 |
| `PRECISION_ACTION_SELECTION` | 100 |
| `EVALUATION_TYPE` | EFE |

Table 4: The value of each hyper-parameter used by $BTAI_{BF}$ in this section. `NB_SIMULATIONS` is the number of simulations run during the experiment. `NB_ACTION_PERCEPTION_CYCLES` is the maximum number of actions executed in each simulation, after which the simulation is terminated. `NB_PLANNING_STEPS` is the number of planning iterations performed by the agent. `EXPLORATION_CONSTANT` is the exploration constant of the UCT criterion. `PRECISION_PRIOR_PREFERENCES` is the precision of the prior preferences. `PRECISION_ACTION_SELECTION` is the precision of the distribution used for action selection. `EVALUATION_TYPE` is the type of cost used to evaluate the node during the tree search. Those hyper-parameters can be used to re-run the experiments using the code of the following GitHub repository: `https://github.com/ChampiB/Branching_Time_Active_Inference`.

As shown in Table **??**, the agent is able to solve: 86.1% of the task when using a granularity of eight, 97.7% of the task when using a granularity of four, and 98.6% of the task when using a granularity of two. However, as the performance improves from 86.1% to 98.6%, the computational time required to run each simulation skyrockets from around 50 milliseconds to around 17.5 seconds. In other words, a simulation with a granularity of two is 350 times slower than a simulation with a granularity of eight.

| Planning iterations | Granularity | P(solved) | Time (ms) |
|---|---|---|---|
| 50 | 8 | 0.861 | 49.93 ± 36.4124 |
| 50 | 4 | 0.977 | 241.63 ± 118.379 |
| 50 | 2 | 0.986 | 17503.8 ± 12882.8 |

Table 5: The percentage of the dSprites environment solved by the $BTAI_{BF}$ agent when using a granularity of eight, four and two. Note, when a granularity of two is used, there are $17 \times 16 \times 3 = 816$ possible states. The last column reports the average execution time required for one simulation and the associated standard deviation. Note, the change in time granularity to milliseconds.

### 3.4 $BTAI_{3MF}$ modeling approach and results

In this section, we evaluate our new approach ($BTAI_{3MF}$) on the dSprites environment. In contrast to what is shown in Figure **??**, $BTAI_{3MF}$ does not observe one index for each possible configuration of shape, and $(X, Y)$ positions. Instead, $BTAI_{3MF}$ has five observed variables representing the shape, the orientation, the scale, as well as the X and Y position, respectively. Each of those observed variable has its hidden state counterparts. Each observation depends on its hidden state counterparts through an identity matrix. This parametrisation is common in the literature on active inference, see (**?**) for an example. The transition mappings of the hidden variables representing the shape, orientation, and scale, are defined as an indentity matrix. This forwards the state value at time $t$ to the next time step $t+1$. For the hidden variables representing the X and Y position of the shape, the transition is set to reflect the dynamics of the dSprites environment when the actions taken are repeated eight times, i.e., if the action "DOWN" is selected, then the agent's position in Y will be decreased by eight before the start of the next action-perception cycle (**?**).

The hyper-parameters used in those simualtions are presented in Table **??**. Note, the hyper-parameters values are the same for all BTAI models presented in this paper. Only the number of action perception cycles, and the number of planning iterations may vary from one experience to the next.

Table **??** shows the results obtained by $BTAI_{3MF}$ on the dSprites environment when running 100 trials. Due to the change in the format of representations, the agent exhibits little increase in execution time as the granularity decreases, however, in general, the capacity to solve the task increases with this reduction in granularity. When a granularity of one is used, the agent is able to solve the task perfectly with 150 planning iterations.

Note, the agent using a granularity of 1 and 150 planning iterations is as fast as the agent using a granularity of 1 and 50 planning iterations. This is because as the number of planning iterations increase the agent requires more computational time per action-perception cycle, but as the agent performance increases on the task, the agent reaches the goal state faster, and therefore requires less action-perception cycles per simulation. To conclude, the agent with 150 planning iterations requires less action-perception cycles per simulation, but more time per action-perception cycle than the agent with 50 planning iterations. The code relevant to this section is available at the following URL: `https://github.com/ChampiB/BTAI_3MF`.

| Name | Value |
|---|---|
| NB_SIMULATIONS | 100 |
| NB_ACTION_PERCEPTION_CYCLES | 50 |
| NB_PLANNING_STEPS | 50 or 100 or 150 |
| EXPLORATION_CONSTANT | 2.4 |
| PRECISION_PRIOR_PREFERENCES | 1 |
| EVALUATION_TYPE | EFE |

Table 6: The value of each hyper-parameter used by $BTAI_{3MF}$ in this section. NB_SIMULATIONS is the number of simulations run during the experiment. NB_ACTION_PERCEPTION_CYCLES is the maximum number of actions executed in each simulation, after which the simulation is terminated. NB_PLANNING_STEPS is the number of planning iterations performed by the agent. EXPLORATION_CONSTANT is the exploration constant of the UCT criterion. PRECISION_PRIOR_PREFERENCES is the precision of the prior preferences. EVALUATION_TYPE is the type of cost used to evaluate the node during the tree search. Those hyper-parameters can be used to re-run the experiments using the code of the following GitHub repository: https://github.com/ChampiB/BTAI_3MF.

| Planning iterations | Granularity | P(solved) | Time (sec) |
|---|---|---|---|
| 50 | 8 | 0.895 | $1.279 \pm 12.8$ |
| 50 | 4 | 0.977 | $1.279 \pm 12.8$ |
| 50 | 2 | 0.996 | $1.279 \pm 12.8$ |
| 50 | 1 | 0.72 | $2.559 \pm 18.01$ |
| 100 | 1 | 0.77 | $5.119 \pm 25.209$ |
| 150 | 1 | 1 | $2.559 \pm 18.01$ |

Table 7: This table presents the percentage of the dSprites environment solved by the $BTAI_{3MF}$ agent when using a granularity of eight, four, two and one. Note, when a granularity of one is used, there are $33 \times 32 \times 3 \times 40 \times 6 = 760,320$ possible state configurations. The last column reports the average execution time required of one simulation and the associated standard deviation.

## 4. Conclusion

In this paper, we presented a new version of Branching Time Active Inference that allows for modelling of several observed and latent variables. Taken together, those variables constitute a temporal slice. Within a slice, the model is equipped with prior beliefs over the initial latent variables, and each observation depends on a subset of the latent variables through the likelihood mapping. Additionally, the latent states evolve over time according to the transition mapping that describes how each latent variable at time $t+1$ is generated from a subset of the hidden states at time $t$ and the action taken.

At the beginning of each trial, the agent makes an observation for each observed variable, and computes the posterior over the latent variables using belief propagation. Then, a Monte-Carlo tree search is performed to explore the space of possible policies. During the tree search, each planning iteration starts by selecting a node to expand using the UCT criterion. Then, the children of the selected node are expanded, i.e., one child per action. Next, the posterior over the latent variables of the expanded nodes is computed by performing forward predictions using the known transition mapping, and the posterior beliefs over the latent states of the node selected for expansion. Once the posterior is computed, the expected free energy can be computed and back-propagated through the tree. The planning process stops after reaching a maximum number of iterations.

In the results section, we compared our new approach, called $BTAI_{3MF}$, to two earlier versions of branching time active inference, named $BTAI_{VMP}$ (**??**) and $BTAI_{BF}$ (**?**). Briefly, at the current time step $t$: $BTAI_{VMP}$ performs variational message passing (VMP) with a variational distribution composed of only one factor, $BTAI_{BF}$ performs exact inference using Bayes theorem, and $BTAI_{3MF}$ implements belief propagation to compute the marginal posterior over each latent variable. For the hidden variables in the future, $BTAI_{VMP}$ does the same mean-field approximation

as at time step $t$ and performs VMP, $BTAI_{BF}$ performs Bayesian prediction to compute the posterior over the only latent variable being modelled, and likewise, $BTAI_{3MF}$ performs prediction to compute the posterior over all future latent variables.

Since, none of the aforementioned approaches are equipped with deep neural networks, we compared them on a version of the dSprites environment in which the metadata of the dSprites dataset are used as inputs to the model instead of the dSprites images. The best performance obtained by $BTAI_{VMP}$ was to solve 96.9% of the task in 5.1 seconds. Importantly, $BTAI_{VMP}$ was previously compared to active inference as implemented in SPM both theoretically and experimentally (**??**). $BTAI_{BF}$ was able to solve 98.6% of the task but at the cost of 17.5 seconds of computation. Note, $BTAI_{BF}$ was using a granularity of two (i.e., 816 states) while $BTAI_{VMP}$ was using a granularity of four (i.e., 216 states), which is why $BTAI_{BF}$ seems to be three times slower than $BTAI_{VMP}$. In reality, if $BTAI_{BF}$ had been using a granularity of four, it would have been much faster than $BTAI_{VMP}$ while maintaining a similar performance, i.e., around 96.9% of the task solved. Finally, $BTAI_{3MF}$ outperformed both of its predecessors by solving the task completely (100%, granularity of 1) in only 2.559 seconds. Importantly, $BTAI_{3MF}$ was able to model all the modalities of the dSprites environment for a total of $760,320$ possible states.

In addition to the major boost in performance and computational time, $BTAI_{3MF}$ provides an improved modelling capacity. Indeed, the framework can now handle the modelling of several observed and latent variables, and takes advantage of the factorisation of the generative model to perform inference efficiently. As described in detail in Appendix A, we also provide a high level notational language for the creation of $BTAI_{3MF}$ that aims to make our approach as staightforward as possible to apply to new domains. The high-level notational language allows the user to create models by simply declaring the variables it contains, and the dependencies between those variables. Then, the framework performs the inference process automatically. Moreover, driven by the need for interpretability, we developed a graphical user interface to analyse the behaviour and reasoning of our agent, which is described in Appendix B.

There are two major directions of future research that may be explored to keep scaling up this framework. First, $BTAI_{3MF}$ is not yet equipped with deep neural networks (DNNs), and is therefore unable to handle certain types of inputs, such as images. In addition to the integration of DNNs into the framework, further research should be performed in order to learn useful sequences of actions. Typically, in the current version of $BTAI_{3MF}$, we built in the fact that each action should be repeated eight times in a row. This inductive bias works well in the context of the dSprites environment, but may be a limitation in other contexts.

It is also worth reflecting on how the $BTAI_{3MF}$ model sits with theories of brain function. In this respect, it is interesting to consider neural correlates of the "standard" approach that $BTAI_{3MF}$ is being placed in opposition to. As previously discussed, this standard active inference approach could be considered as monolithically tabular; that is, the key matrices, such as the likelihood mapping (the $\boldsymbol{A}$ matrix) and the transition mapping (the $\boldsymbol{B}$ matrix), grow in size exponentially with the number of states and observations. This is simply due to a combinatorial explosion, e.g. the set of all combinations of states grows intractably with the number of states.

How would the combinations of states in the monolithic tabular approach be represented in the brain? The obvious neural correlate would be conjunctive (binding) neurons (**?**), which become active when multiple feature values are present; for example, one might have a neural unit for every X, Y combination in the dSprites environment. If this is to be realised with a fully localist code, i.e. one unit for every combination, in the absence of any hierarchical structure, the required number of conjunctive units would explode in the same way as the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices do. This is why some models have proposed a binding resource that supports distributed (rather than localist) representations (**?**), which scale more tractably.

$BTAI_{3MF}$ avoids this combinatorial explosion by not combining features, enabling them to be represented separately. In a very basic sense, this separated representation is consistent with the observation that the brain contains distinct, physically separated, feature maps, e.g. **?**. Thus, at least to some extent, different feature dimensions are processed separately in the brain, as they are in $BTAI_{3MF}$.

The time-slice idea in $BTAI_{3MF}$ assumes a kind of discrete synchronising global clock. That is, even though features have been separated from one another and may be considered to execute in different parts of the system, they update in lock-step. That is, implicitly, time is a binder, it determines which values of different feature dimensions/states are associated, e.g. an X-dimension value is associated with a particular Y-dimension value because they

are so assigned in the same temporal slice. In this sense, in $BTAI_{3MF}$, time synchronisation resolves the binding problem.

This aspect of $BTAI_{3MF}$ resonates with theories of binding based upon oscillatory synchrony (**?**). These theories suggest that different feature dimensions are bound by the corresponding neurons firing in synchrony relative to an ongoing oscillation, with that ongoing oscillation potentially playing the role of a global clock. Such oscillatory synchrony can be seen as a way to resolve the binding problem that does not require conjunctive units.

Conjunction error experiments, e.g. **?**, are also relevant here. In these experiments, participants make errors in associating multiple feature dimensions, perceiving illusory percepts, e.g. if a red K is presented before a blue A in a rapid serial visual presentation stream, in some cases, a red A and a blue K is perceived. These experiments firstly, re-emphasize that different feature dimensions are processed separately, as per $BTAI_{3MF}$: if feature dimensions were not separated, then conjunction errors could not happen. Additionally though, these experiments suggest that there is not a "perfect" synchronising global clock, since if there were, there would not be any conjunction errors even despite separation of feature dimensions. Generating such conjunction error patterns is an interesting topic for future $BTAI_{3MF}$ modelling work.


## Acknowledgments

TO BE FILLED


## Appendix A: How to create a $BTAI_{3MF}$ agent?

In this appendix, we describe how to build a $BTAI_{3MF}$ agent using our framework. The relevant code can be found in the file `main_BTAI_3MF.py` at the following URL: `https://github.com/ChampiB/BTAI_3MF`. Any script running a $BTAI_{3MF}$ agent must start by instantiating an environment in which the agent will be run. Our code provides an implementation of the dSprites environment, which can be created as follow:

```
# Create the environment.
env = dSpritesEnv(granularity=1, repeat=8)
env = dSpritesPreProcessingWrapper(env)
```

The first line creates the dSprites environment, the second makes sure that the observations generated by the environment are in the format expected by the agent. Once the environment has been created, we need to define the parameters of the model. Assume that we want to have a latent variable $S_t^{shape}$ representing the shape in the current image. This variable can takes three values, i.e., zero for squares, one for ellipses and two for hearts. In this case, the parameters of the prior over $S_t^{shape}$ may be created as:

```
# Create the parameters of the prior over the latent variable shape.
d = {}
d["S_shape"] = torch.tensor([0.2, 0.3, 0.5])
```

The first line above creates a python dictionary, the second line adds a vector of parameters in the dictionary. This vector can be accessed using the key "S_shape", which corresponds to the name of the latent variable. The values in d["S_shape"] mean that a priori the agent believes it will observe a square with probability 0.2, an ellipse with probability 0.3, and a heart with probability 0.5. Also, by convention, the name of a latent variable must start with "S_". Similarly, if we assume that the shape is provided to the agent through an observed variable $O_t^{shape}$, we can create the parameters of the likelihood mapping for this variable as:

```
# Create the parameters of the likelihood mapping for the shape variable.
a = {}
a["O_shape"] = torch.eye(3)
```

The first line above creates a python dictionary, and the second line adds a 3×3 identity matrix[1] in the dictionary. This reflects the fact that there is a one-to-one relationship between the value taken by $S_t^{shape}$ and $O_t^{shape}$. Also, by

1. Note, in practice the identity matrix is noisy to avoid taking the logarithm of zero.

convention, the observations name must start with "O_". Since, defining all the parameters manually can be tedious, our framework provides built-in functions that return the model parameters for the dSprites environment. Using those functions, the parameters can be retrieved as follows:

```
# Define the parameters of the generative model.
a = env.a()
b = env.b()
c = env.c()
d = env.d(uniform=True)
```

Once all the parameters have been created, it is time to define the structure of the generative model. This can be done using a temporal slice builder, which is an object used to facilitate the creation of a temporal slice. First, we need to create the builder as follows:

```
# Create the temporal slice builder.
ts_builder = TemporalSliceBuilder("A_1", env.n_actions)
```

The builder takes two parameters, i.e., the name of the action random variable (i.e., "A_1") that must start by "A_", and the number of possible actions (i.e., env.n_actions = 4). Then, we need to tell the builder what state variables should be created, and what are the parameters of the prior beliefs over those variables. For the dSprites environment, this can be done as follows:

```
# Add the latent states of the model to the temporal slice.
ts_builder.add_state("S_pos_x", d["S_pos_x"]) \
    .add_state("S_pos_y", d["S_pos_y"]) \
    .add_state("S_shape", d["S_shape"]) \
    .add_state("S_scale", d["S_scale"]) \
    .add_state("S_orientation", d["S_orientation"])
```

The function "add_state" adds a state variable to the temporal slice. The first parameter of this function is the name of the state to be added, and the second argument is the parameters of the prior beliefs over this new state. Next, we need to add the variables corresponding to the observations made by the agent. For the dSprites environment, this can be done as follows:

```
# Define the likelihood mapping of the temporal slice.
ts_builder.add_observation("O_pos_x", a["O_pos_x"], ["S_pos_x"]) \
    .add_observation("O_pos_y", a["O_pos_y"], ["S_pos_y"]) \
    .add_observation("O_shape", a["O_shape"], ["S_shape"]) \
    .add_observation("O_scale", a["O_scale"], ["S_scale"]) \
    .add_observation("O_orientation", a["O_orientation"], ["S_orientation"])
```

The function "add_observation" adds an observation variable to the temporal slice. The first parameter of this function is the name of the observation to be added, the second argument is the parameters of the likelihood mapping for this new observation, and the third parameter is the list of parents on which the observation depends. The next step is the definition of the transition mapping for each hidden state, which can be performed as follows:

```
# Define the transition mapping of the temporal slice.
ts_builder.add_transition("S_pos_x", b["S_pos_x"], ["S_pos_x", "A_1"]) \
    .add_transition("S_pos_y", b["S_pos_y"], ["S_pos_y", "A_1"]) \
    .add_transition("S_shape", b["S_shape"], ["S_shape"]) \
    .add_transition("S_scale", b["S_scale"], ["S_scale"]) \
    .add_transition("S_orientation", b["S_orientation"], ["S_orientation"])
```

The function "add_transition" adds a transition mapping to the temporal slice. The first parameter of this function is the name of the state for which the transition is defined, the second argument is the parameters of the transition

mapping for this state, and the third parameter is the list of parents on which the state depends. Importantly, in the above snippet of code, only the states representing the position in x and y of the shape depends on the action variable "A_1". The final step is about the defintion of the prior preferences of the agent, and can be done as follows:

```
# Define the prior preferences of the temporal slice.
ts_builder.add_preference(["O_pos_x", "O_pos_y", "O_shape"], c["O_shape_pos_x_y"])
```

The function "add_preference" adds some prior preferences to the temporal slice. The first parameter of this function is the list of observations for which the prior preferences are defined, and the second argument are the parameters of the prior preferences for those observations. At this stage, the initial temporal slice can be built:

```
# Create the initial temporal slice.
ts = ts_builder.build()
```

Once the initial temporal slice has been created, it is possible to instantiate the agent and implement the action-perception cycle as follows:

```
# Create the agent.
agent = BTAI_3MF(ts, max_planning_steps=150, exp_const=2.4)

# Implement the action-perception cycles.
n_trials = 100
for i in range(n_trials):
    obs = env.reset()
    env.render()
    agent.reset(obs)
    while not env.done():
        action = agent.step()
        obs = env.execute(action)
        env.render()
        agent.update(action, obs)
```

Most of the above code is self explanatory. Put simply, this code runs "n_trials" simulations of the dSprites environment. The line "action = agent.step()" performs inference, planning and action selection. The line "obs = env.execute(action)" executes the selected action in the environment, and the line "agent.update(action, obs)" updates the agent so that it has taken into account the action taken in the environment and the observations received.

## Appendix B: How to inspect a $BTAI_{3MF}$ agent?

In this appendix, we describe how to analyse a $BTAI_{3MF}$ agent using our graphical user interface (GUI). The relevant code can be found in the file `analysis_BTAI_3MF.py` at the following URL: `https://github.com/ChampiB/BTAI_3MF`. The first step is to create the environment and agent as described in Appendix A. Then, we create a GUI object and run the main loop as follows:

```
    # Create the GUI for analysis.
    gui = GUI(env, agent)
    gui.loop()
```

The above two lines should open a graphical user interface as shown in Figure **??**. When clicking on the node of the current temporal slice $TS(t)$, one can obtain additional information about this temporal slice, c.f., Figure **??**. When clicking on the button named "Next planning iteration" in Figure **??**, a planning iteration is performed and the tree displayed on the right-hand-side of this frame is updated as shown in Figure **??**. When clicking on the root's children, e.g., "TS(1)", it is possible to navigate through the tree created by the MCTS algorithm as shown in Figure **??**. When "TS(1)" is displayed as the new root as in Figure **??**, clicking on "TS(1)" again will display the information

of this node as depicted by Figure **??**. Finally, Figure **??** shows how the ambiguity term of the expected free energy can be decomposed into its component parts.

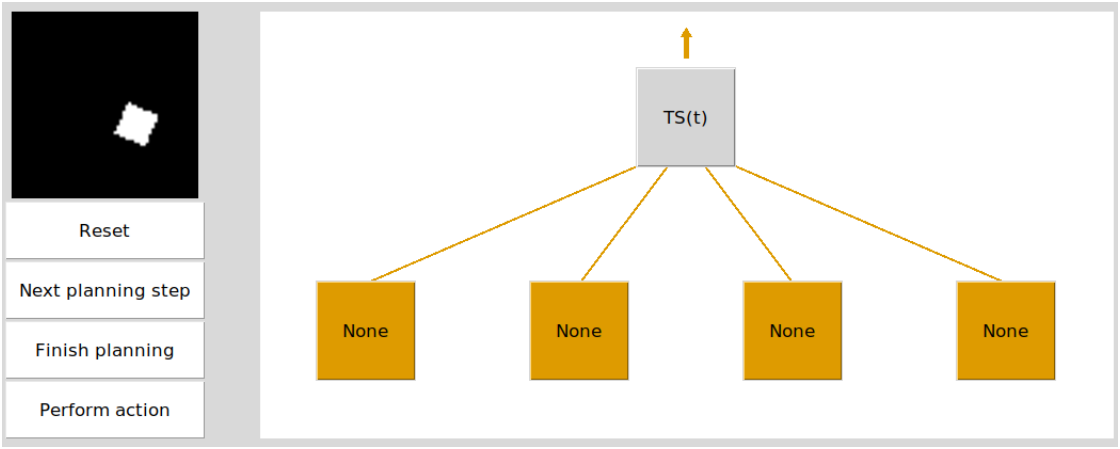



Figure 6: This figure illustrates the visualisation frame of the GUI used to analyse a $BTAI_{3MF}$ agent. The image corresponding to the current state of the environment is displayed in the upper-left corner. Under the image are four buttons allowing the user to: reset the environment and agent, perform the next planning iteration, perform all the remaining planning iterations, and perform the current best action in the environment. Finally, on the right hand side of the image is a depiction of the MCTS planning, where $TS(t)$ represents the current temporal slice. At the moment, the current temporal slice has no children, and therefore its children are displayed in orange with the text "None". Additionally, the current slice has no parent because it is the tree's root. Therefore, the arrow above the $TS(t)$ node is also orange.

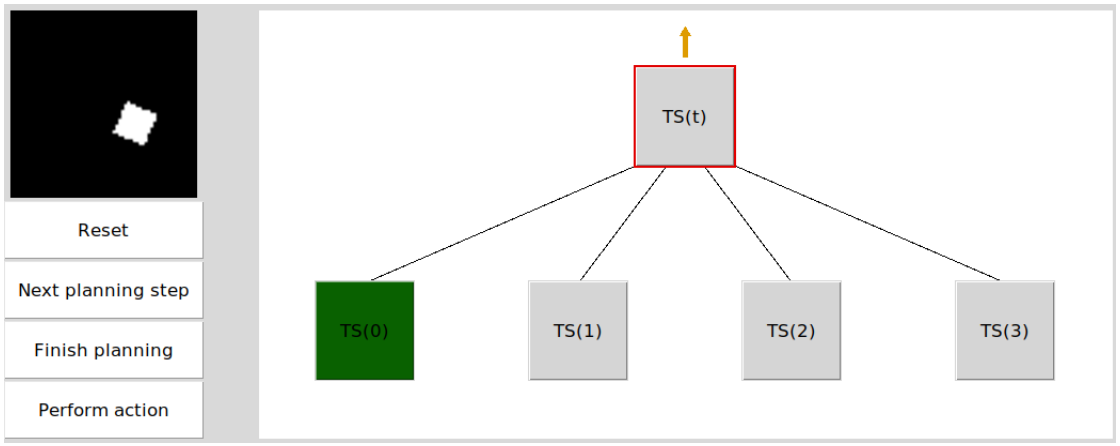



Figure 7: This figure illustrates the visualisation frame of the GUI used to analyse a $BTAI_{3MF}$ agent after performing one planning iteration. The children of the root node are now available. One of them is displayed in green, it corresponds to the best action found so far by the MCTS algorithm. The root node has a red square surronding it, which means that it was selected for expansion by the MCTS algorithm.

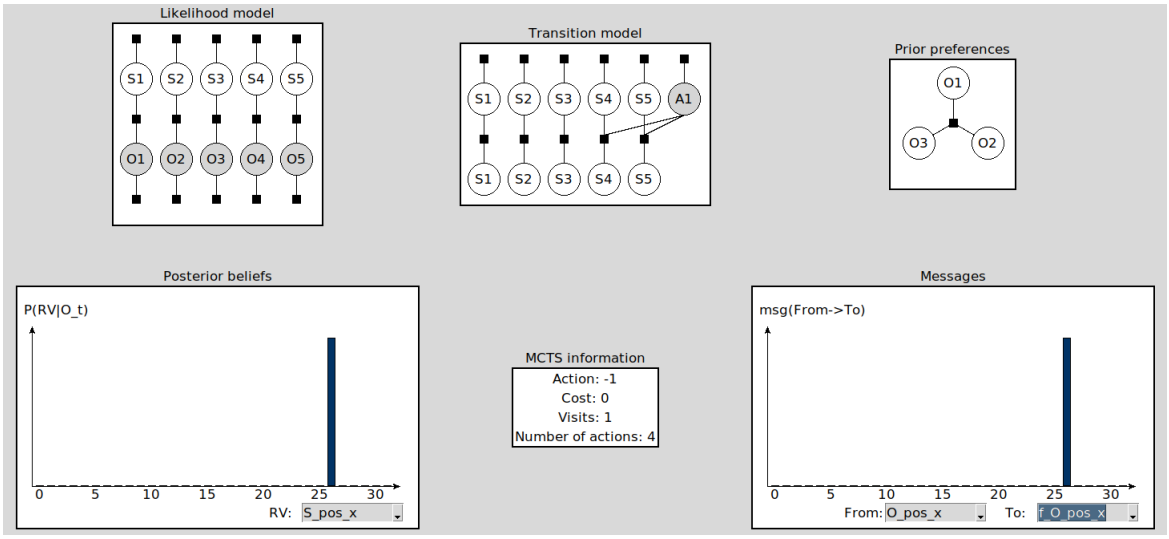



Figure 8: This figure illustrates the frame displaying the information of the current temporal slice of the $BTAI_{3MF}$ agent. Six widgets are displayed. The first displays the structure of the likelihood model using the factor graph formalism. On this graph, we see that the model is composed of five obervations and five hidden states. Each observation depends on only one hidden state. The second widget displays the structure of the transition mapping. We see that only two hidden states depend on the action taken by the agent, i.e., the hidden states corresponding to the X and Y position of the shape. The third widget shows the structure of the prior preferences. Here, there is only one factor over three random variables, i.e., the shape and its (X, Y) position. Note, when moving your mouse over a variable in the likelihood, transition or prior preference widget the complete name of the variable is displayed, e.g., when moving over "S1" the label "S_shape" is displayed. The fourth widget illustrates the posterior over the latent variable corresponding to the x position of the shape. The random variable whose posterior is displayed can be changed either by using the combo box in the bottom-right corner of the widget or by clicking on a latent variable in the likelihood model widget. The fifth widget displays information related to the Monte-Carlo tree search. Finally, the last widget illustrates the message sent from the observation variable corresponding to the X position of the shape to its likelihood factor.

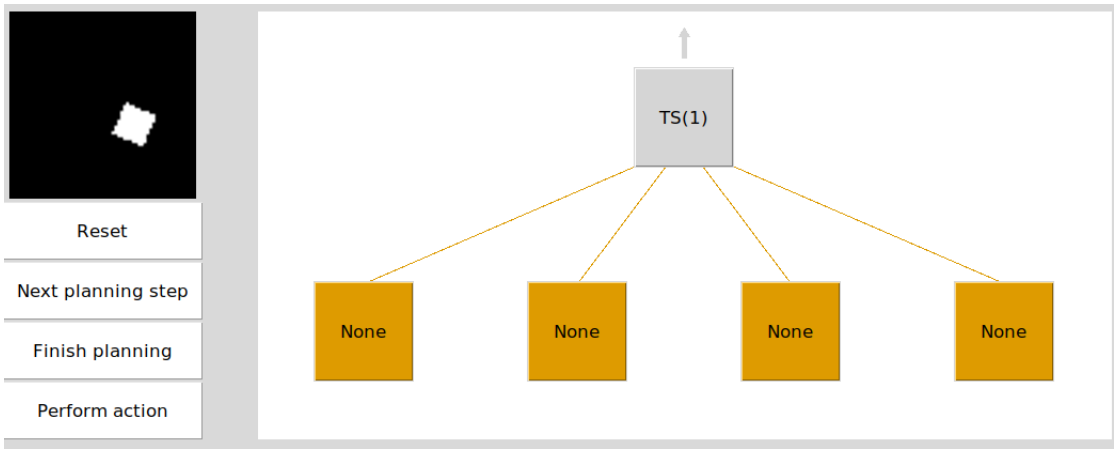



Figure 9: This figure illustrates what happens when clicking on the child "TS(1)" in Figure **??**. Put simply, "TS(1)" becomes the new root and we see that its children have not been expanded yet. Additionally, the arrow above the "TS(1)" node is gray meaning that this node has a parent, i.e., "TS(t)". Clicking on this arrow leads us back to Figure **??**.

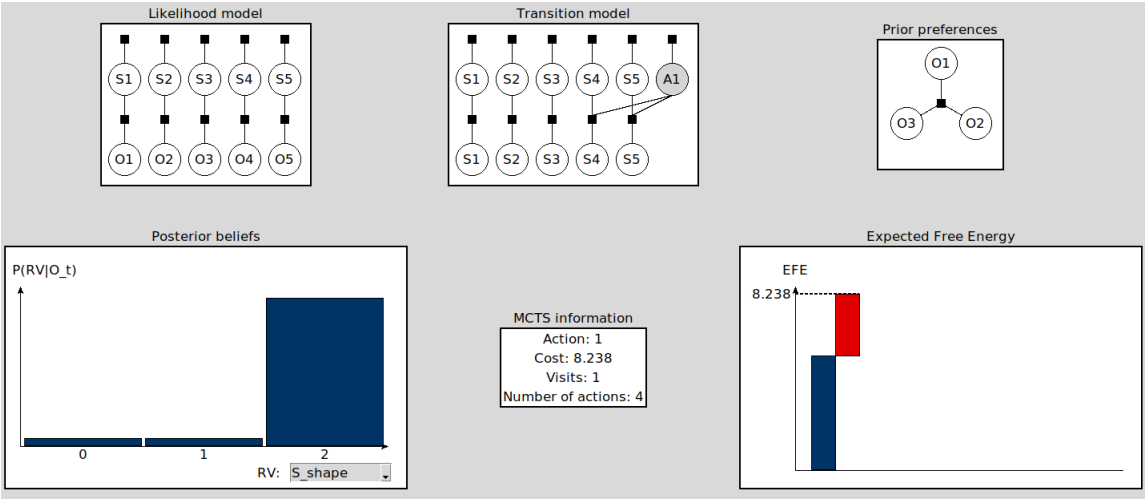



Figure 10: This figure illustrates what happens when clicking on "TS(1)" in Figure **??**. Most of the widgets have already been explained with the exception of the one in the bottom right-corner, which displays how the expected free energy decomposes into risk (blue box) and ambiguity (red box). When clicking on the blue or red box, the decomposition of the risk or ambiguity term is displayed as shown in Figure **??**.

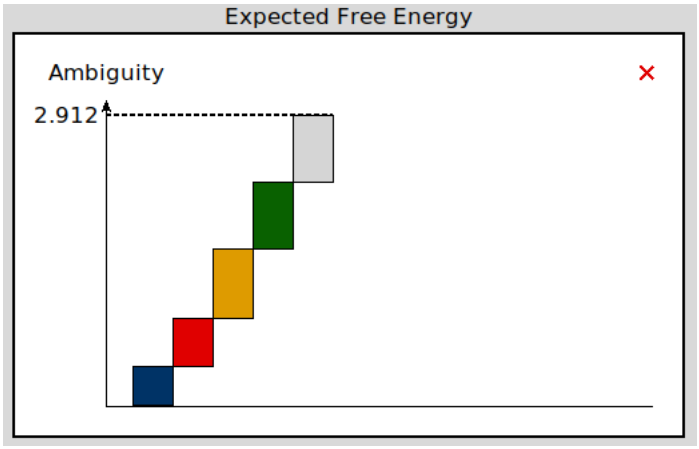



Figure 11: This figure illustrates how the ambiguity term decomposes into the ambiguity of the likelihood of each observed variable, i.e., the ambiguity of "O_shape" in blue, "O_scale" in red, "O_orientation" in orange, "O_pos_x" in green, and "O_pos_y" in gray.

## Appendix C: sum-rule, product-rule and d-separation criterion.

In this appendix, we explain three important properties than are used in the core of the paper, namely: the sum-rule and product-rule of probability and the d-separation criterion.

**Sum-rule of probability**

Given a set of random variables $X = \{X_1, ..., X_n\}$, and a joint distribution $P(X_1, ..., X_n)$ over $X$. The sum-rule allows to sum out a subset of the random variables. Here are a few examples:

$$P(X_1, ..., X_{n-1}) = \sum_{X_n} P(X_1, ..., X_n),$$
$$P(X_1, ..., X_{n-2}) = \sum_{X_{n-1}} \sum_{X_n} P(X_1, ..., X_n),$$
$$P(X_1, ..., X_{n-3}) = \sum_{X_{n-2}} \sum_{X_{n-1}} \sum_{X_n} P(X_1, ..., X_n).$$

Note, the sum-rule can also be used with a conditional distribution $P(X_1, ..., X_n|Y_1, ..., Y_m)$, for examples:

$$P(X_1, ..., X_{n-1}|Y_1, ..., Y_m) = \sum_{X_n} P(X_1, ..., X_n|Y_1, ..., Y_m),$$
$$P(X_1, ..., X_{n-2}|Y_1, ..., Y_m) = \sum_{X_{n-1}} \sum_{X_n} P(X_1, ..., X_n|Y_1, ..., Y_m),$$
$$P(X_1, ..., X_{n-3}|Y_1, ..., Y_m) = \sum_{X_{n-2}} \sum_{X_{n-1}} \sum_{X_n} P(X_1, ..., X_n|Y_1, ..., Y_m).$$

**Product-rule of probability**

Given a set of random variables $X = \{X_0, ..., X_n\}$, and a joint distribution $P(X_0, ..., X_n)$ over $X$. The product-rule allows us to factorise the joint into a product of factors without doing any conditional independence assumptions about $P(X_1, ..., X_n)$. More formally:

$$P(X_0, ..., X_n) = P(X_n) \prod_{i=0}^{n-1} P(X_i|X_{i+1:n}),$$

where $X_{i:j} = \{X_i, ..., X_j\}$ is the set of random variables containing all the variables between $X_i$ and $X_j$ (included). Note, the product-rule can also be used with a conditional distribution $P(X_0, ..., X_n|Y_1, ..., Y_m)$:

$$P(X_0, ..., X_n|Y_1, ..., Y_m) = P(X_n|Y_1, ..., Y_m) \prod_{i=0}^{n-1} P(X_i|X_{i+1:n}, Y_1, ..., Y_m).$$

**The d-separation criterion**

The d-separation criterion is a tool than can be used to check whether two sets of random variables ($X$ and $Y$) are independent given a third set of random variables $Z$. More formally, the d-separation criterion is a tool to check whether $X \perp\!\!\!\perp Y \mid Z$. Knowing that $X \perp\!\!\!\perp Y \mid Z$ holds in a distribution $P$ is useful because if $X \perp\!\!\!\perp Y \mid Z$, then:

$$\begin{aligned} P(X, Y|Z) &= P(X|Y, Z)P(Y|Z) && \text{(product-rule)} \\ &= P(X|Z)P(Y|Z). && (X \perp\!\!\!\perp Y \mid Z) \end{aligned}$$

First, let $G = (\mathcal{X}, \mathcal{E})$ be a graph over a set of nodes $\mathcal{X}$ connected by a set of directed edges $\mathcal{E}$. Given two nodes in the graph (i.e., $N_i, N_j \in \mathcal{X}$), we note: (i) $N_i \to N_j$ if there is a directed edge from $N_i$ to $N_j$ in the graph, (ii) $N_i \leftarrow N_j$ if the graph contains a directed edge from $N_j$ to $N_i$, and (iii) $N_i \rightleftarrows N_j$ if (i) or (ii) holds. Second, we say that there is a *trail* between two nodes (i.e., $N_1, N_n$) in the graph, if there is a sequence of distinct nodes $N = (N_1, ..., N_n)$, such that: $N_i \rightleftarrows N_{i+1}$ holds for all $i \in \{1, ..., n-1\}$. Third, we say that a trail between $N_1$ and $N_n$ is *active* if: (a) each time there is a v-structure (i.e., $N_{i-1} \to N_i \leftarrow N_{i+1}$) in the trail, then either $N_i$ or (at least) one of its descendants are in $Z$, and (b) no other node along the trail are in $Z$. Finally, we say that $X$ and $Y$ are *d-separated* by $Z$ if for all $X_i \in X$ and $Y_i \in Y$ there is no active trail between $X_i$ and $Y_i$ (given $Z$).

Using our terminology, the d-separation criterion states that if $X$ and $Y$ are d-separated by $Z$ in a graph $G$ representing the factorisation of a distribution $P$, then $X \perp\!\!\!\perp Y \,|\, Z$ holds in the distribution $P$. Intuitively, the d-separation criterion help us to determine whether $X \perp\!\!\!\perp Y \,|\, Z$ holds in $P$ by looking at the topology of the graph $G$. For example, consider the Bayesian network illustrated in Figure **??**, and let $P$ be the joint distribution represented by this Bayesian network. Using the product rule, we get:

$$P(A, B, C, D, E, F) = P(F|A, B, C, D, E)P(E|A, B, C, D)P(C|A, B, D)P(D|A, B)P(B|A)P(A).$$

Note, that all trails between $C$ and $A, D$ are blocked by $B$, i.e., there is no active trails between $C$ and $A, D$ given $B$. Thus, we have $C \perp\!\!\!\perp A, D \,|\, B$ and:

$$P(A, B, C, D, E, F) = P(F|A, B, C, D, E)P(E|A, B, C, D)\boldsymbol{P(C|B)}P(D|A, B)P(B|A)P(A).$$

Moreover, there is no active trail between $B$ and $A$ given $\emptyset$, therefore $B \perp\!\!\!\perp A \,|\, \emptyset$ and:

$$P(A, B, C, D, E, F) = P(F|A, B, C, D, E)P(E|A, B, C, D)P(C|B)P(D|A, B)\boldsymbol{P(B)}P(A).$$

Using the same reasoning, one can see that $F \perp\!\!\!\perp A, B, C, D \,|\, E$ and thus:

$$P(A, B, C, D, E, F) = \boldsymbol{P(F|E)}P(E|A, B, C, D)P(C|B)P(D|A, B)P(B)P(A).$$

Finally, using the d-separation one more time leads to the following factorisation for $P$:

$$P(A, B, C, D, E, F) = P(F|E)\boldsymbol{P(E|B, D)}P(C|B)P(D|A, B)P(B)P(A).$$

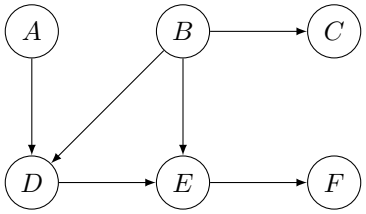



Figure 12: This figure illustrates a Bayesian network in which the following independences assumptions **hold**: $A \perp\!\!\!\perp B \,|\, \emptyset$; $A, D \perp\!\!\!\perp C \,|\, B$; and $A \perp\!\!\!\perp E \,|\, D, B, C$. In contrast, the following independences assumptions **does not hold**: $A \perp\!\!\!\perp B \,|\, D$; $A \perp\!\!\!\perp E \,|\, B, C$; and $A \perp\!\!\!\perp B \,|\, E$ .