Interpretable Preference-based Reinforcement Learning with Tree-Structured Reward Functions

Tom Bewley
University of Bristol
Bristol, United Kingdom
tom.bewley@bristol.ac.uk

Freddy Lecue
CortAIx, Thales
Montréal, Canada
freddy.lecue@inria.fr

ABSTRACT

The potential of reinforcement learning (RL) to deliver aligned and performant agents is partially bottlenecked by the reward engineering problem. One alternative to heuristic trial-and-error is preference-based RL (PbRL), where a reward function is inferred from sparse human feedback. However, prior PbRL methods lack interpretability of the learned reward structure, which hampers the ability to assess robustness and alignment. We propose an online, active preference learning algorithm that constructs reward functions with the intrinsically interpretable, compositional structure of a tree. Using both synthetic and human-provided feedback, we demonstrate sample-efficient learning of tree-structured reward functions in several environments, then harness the enhanced interpretability to explore and debug for alignment.

KEYWORDS

Interactive RL; Preference models; Interpretability; Decision trees

ACM Reference Format:
Tom Bewley and Freddy Lecue. 2022. Interpretable Preference-based Reinforcement Learning with Tree-Structured Reward Functions. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022*, IFAAMAS, 9 pages.

1 INTRODUCTION

It has recently been argued that the paradigm of reinforcement learning (RL) , in which agents learn action-selection policies by exploration to maximise future reward, is sufficiently general to give rise to most, if not all, aspects of natural and artificial intelligence [24]. However, the origin of the reward signal itself has received limited research attention compared with the algorithms used to optimise it. Singh et al. [25] argue that the traditional presentation of reward, as a known and hard-wired property of an agent's environment, is misleading with respect to both biological plausibility and real-world technical applications. Motivated by evolutionary biology, they propose a differentiation between a latent fitness function F, which produces a scalar evaluation of the true quality of a behavioural trajectory, and the reward function R, which is a mere proxy to convert global pressures on fitness into local pressures on immediate decision-making. R may have a nontrivial relationship to F because it must be tailored to both the learning dynamics of the agent and the structure of the environment. During the learning process, two nested optimisation loops are at work: inner adaptation of the RL agent within a given reward structure R, and outer adaptation of R to better align it with fitness.

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online. © 2022 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

As RL becomes more powerful, the engineering of aligned reward functions will become "both more important and more difficult" [10]. The prevailing reliance on heuristic approaches, which already presents challenges to real-world deployment [19] and hampers the use of RL non-experts [28], is likely to prove inadequate in the face of vastly more capable agents which can exploit any specification errors. Several alternatives to heuristic reward engineering have been proposed, including inverse RL [1], inverse reward design [15], and advice-taking mechanisms [20], all of which involve interaction with a human with (possibly tacit) knowledge of F. In this paper, we focus on yet another human-in-the-loop approach, *preference-based RL* (PbRL) [28], in which a reward function is inferred from preferences expressed by the human over sets of candidate behaviours, indicating which have higher fitness. Given a dataset of preference labels, R is constructed to reward commonly-preferred states and actions, and this function is used to train the agent. While ostensibly restrictive, preference feedback places low cognitive demands on the human, requires little domain expertise or training, and tends to yield lower variance than direct fitness labelling since it does not require the human to fix an absolute scale [14].

The outcome of PbRL is a policy whose true fitness depends on its expected reward under R, and the alignment of R to F. Without ground-truth knowledge of F, it is hard to define quantitative metrics for the latter, which instead becomes a fuzzy, multi-faceted judgement, requiring the assessor to build an intuitive understanding of the structure of R and its effect on agent learning. In large part, this is an *interpretability* problem. The importance of interpretability for human-in-the-loop RL has been highlighted in surveys [3, 19], and some post hoc analysis has been applied to learnt reward functions to gain some insight into feature influence [21, 23], but to our knowledge, there have been no efforts to make R *intrinsically* interpretable (loosely speaking, human-readable) by constraining its functional form. Indeed, prior works implement reward functions as deep neural networks, or ensembles thereof [8], which are notoriously opaque to scrutiny.

We present a PbRL algorithm that learns intrinsically interpretable reward functions from human preferences. Specifically, it yields *tree-structured* reward functions, formed of independent components associated with disjoint subsets of the state-action space, and defined hierarchically as a binary tree. The tree is incrementally refined as new preference labels arrive, and the traceability of these changes provides a powerful mechanism for monitoring and debugging. Trees afford both diagrammatic and geometric visualisation, textual description as a rule set in disjunctive normal form, and the efficient computation of feature importance metrics. Maintaining an explicit uncertainty estimate for each reward component also facilitates active preference learning based on upper confidence bounds. We evaluate our algorithm on four benchmark RL problems using both synthetic and human feedback, and in both offline and online learning settings. We observe aligned and sample-efficient learning of tree-structured reward functions in each of these contexts, alongside some informative failure cases. We then harness the enhanced interpretability to explore and debug for alignment.

This paper is structured as follows. Section 2 reviews the general PbRL problem definition, section 3 introduces the additional tree-structuring constraint that we apply to improve interpretability, and section 4 presents our algorithm for approximately solving the constrained problem. Section 5 gives experimental results with a focus on quantitative performance metrics, while section 6 uses case studies to demonstrate the qualitative interpretability benefits of the tree structure for the purpose of alignment. Finally, section 7 briefly concludes and discusses directions for future work.

2 PBRL PROBLEM DEFINITION

The PbRL problem is formalised within a Markov Decision Process without reward (MDP\R) [1], in which at discrete time t, an agent's action $a_t \in \mathcal{A}$ influences the evolution of an environmental state $s_t \in \mathcal{S}$ according to a Markovian dynamics function $P(s_{t+1}|s_t, a_t)$. We specifically consider fixed-length episodic MDP\Rs, in which t is initially 0, s_0 is sampled from an initial distribution P_0, and the process deterministically terminates at a fixed $t = T$. The events of an episode are described by a trajectory in state-action space, $\tau = ((s_0, a_0), ..., (s_{T-1}, a_{T-1})) \in (\mathcal{S} \times \mathcal{A})^T$. There also exists a human observer, who evaluates the quality of trajectories according to the latent fitness function $F : (\mathcal{S} \times \mathcal{A})^T \rightarrow \mathbb{R}$. The ultimate goal of the agent is to learn an action selection policy $\pi(a_t|s_t)$ that maximises the expected fitness over induced trajectories:

$$\underset{\pi}{\operatorname{argmax}}\ \mathbb{E}_{\tau_i \sim \Pr(\tau|P_0, D, \pi)} F(\tau_i). \tag{1}$$

In order to learn about F, the agent must interact with the human. In PbRL, we assume the human cannot specify the analytical form of F, or even evaluate it absolutely for a given trajectory, but can only assess the *relative* fitness of a trajectory pair τ_i, τ_j and provide a label $y_{ij} \in [\varepsilon, 1-\varepsilon]$ indicating their assessment of the probability that τ_i has higher fitness than τ_j (denoted by $\tau_i \succ \tau_j$). Here, $\varepsilon \in (0, 0.5]$ is a noise parameter preventing extreme probabilities. Agent-human interaction therefore consists of sampling trajectory pairs from a distribution $\psi : ((\mathcal{S} \times \mathcal{A})^T)^2 \rightarrow [0, 1]$ and obtaining preference labels in response. The inference of F reduces to minimising some loss ℓ over labelled pairs:

$$\underset{F}{\operatorname{argmin}}\ \mathbb{E}_{(\tau_i,\tau_j)\sim\psi}\ \ell(y_{ij}, H(\tau_i \succ \tau_j|F)). \tag{2}$$

Here, H is a statistical model of human preference labels given estimated fitness values for a trajectory pair, whose definition is an assumption of the modelling process. Adopting a popular hypothesis from the psychological literature, we assume that the probability of the human preferring τ_i to τ_j is a monotonically non-decreasing function of the fitness difference $F(\tau_i) - F(\tau_j)$. Specifically, we adopt Thurstone's law of comparative judgement [27], which models the fitness of a set of n trajectories $F(\tau_1), ..., F(\tau_n)$ as a multivariate normal distribution with mean $\boldsymbol{\mu} \in \mathbb{R}^n$ and covariance $\Sigma \in \mathbb{R}^{n \times n}$.[1]

This leads to the following preference model:

$$H(\tau_i \succ \tau_j|\boldsymbol{\mu}, \Sigma) = \Phi\left(\frac{\boldsymbol{\mu}_i - \boldsymbol{\mu}_j}{\sqrt{\Sigma_{ii} + \Sigma_{jj} - 2\Sigma_{ij}}}\right), \tag{3}$$

where Φ is the standard normal cumulative distribution (CDF).

Following the classic approach of Mosteller [22], we note that under Thurstone's model, a preference label y_{ij} implies that the variance-scaled difference in fitness between τ_i and τ_j is proportional to $\Phi^{-1}(y_{ij})$, where Φ^{-1} is the inverse normal CDF.[2] Therefore, a suitable choice for the labelling loss ℓ is the squared error in this variance-scaled fitness difference. Equation 2 can be rewritten as

$$\underset{\boldsymbol{\mu},\Sigma}{\operatorname{argmin}}\ \mathbb{E}_{(\tau_i,\tau_j)\sim\psi}\left[\Phi^{-1}(y_{ij}) - \frac{\boldsymbol{\mu}_i - \boldsymbol{\mu}_j}{\sqrt{\Sigma_{ii} + \Sigma_{jj} - 2\Sigma_{ij}}}\right]^2. \tag{4}$$

For the agent to perform the optimisation in equation 1, $\boldsymbol{\mu}$ and Σ must be parameterised in a way that generalises to unlabelled trajectories. We adopt a linear model $\boldsymbol{\mu}_i = \mathbf{r}^\top \mathbf{n}_i$, $\Sigma_{ij} = \mathbf{n}_i^\top C \mathbf{n}_j$, $\forall i, j \in \{1..n\}$, where $\mathbf{n}_i \in \mathbb{R}^m$ is a feature vector summarising the trajectory, $\mathbf{r} \in \mathbb{R}^m$ is a vector of weights, and $C \in \mathbb{R}^{m \times m}$ is a covariance matrix associated with $\mathbf{r}$. As in several prior works [1, 2, 8], we add a second level of decomposition by defining $\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^m$ as a function that constructs feature vectors from *individual state-action pairs*, and $\mathbf{n}_i = \sum_{t=0}^{T-1} \phi(s_{it}, a_{it})$ as the unweighted feature expectation over τ_i.[3] By the linearity of the normal distribution, it follows that $\boldsymbol{\mu}_i - \boldsymbol{\mu}_j = \mathbf{r}^\top(\mathbf{n}_i - \mathbf{n}_j)$ and $\Sigma_{ii} + \Sigma_{jj} - 2\Sigma_{ij} = (\mathbf{n}_i - \mathbf{n}_j)^\top C(\mathbf{n}_i - \mathbf{n}_j)$. The final form of equation 2 is thus

$$\underset{\phi,\mathbf{r},C}{\operatorname{argmin}}\ \mathbb{E}_{(\tau_i,\tau_j)\sim\psi}\left[\Phi^{-1}(y_{ij}) - \frac{\mathbf{r}^\top(\mathbf{n}_i - \mathbf{n}_j)}{\sqrt{(\mathbf{n}_i - \mathbf{n}_j)^\top C(\mathbf{n}_i - \mathbf{n}_j)}}\right]^2. \tag{5}$$

The decomposition also allows us to rewrite equation 1 as

$$\underset{\pi}{\operatorname{argmax}}\ \mathbb{E}_{\tau_i \sim \Pr(\tau|P_0, D, \pi)} \sum_{t=0}^{T-1} \mathbf{r}^\top \phi(s_{it}, a_{it}), \tag{6}$$

which is structurally identical to the conventional RL objective of maximising (undiscounted) return in an MDP with reward. Thus, once the agent has inferred the function ϕ and vector $\mathbf{r}$ it can define a reward function $R(s, a) = \mathbf{r}^\top \phi(s, a)$, then employ any unmodified RL algorithm to learn a policy π. As alluded to in the Introduction, R thereby serves as a proxy for F, with the true fitness of π being a function of both its expected return under R (equation 6) and the alignment of R to F, which is approximated by the labelling loss (equation 5). Given that equation 6 is the domain of standard RL, the contribution of this paper is an algorithm for approximately solving equation 5, subject to the particular interpretability constraint outlined in the following section.

3 INTERPRETABILITY CONSTRAINT

We now introduce the key assumption that differentiates our approach from prior work and enables interpretability. That is, we constrain the feature function ϕ so that for all $(s, a) \in \mathcal{S} \times \mathcal{A}$, $\phi(s, a)$ is a one-hot vector. This effectively induces a partition of

[1]Prior works [8, 29] have adopted the Bradley-Terry model [5] which maintains no covariance estimate and uses a logistic function $H(\tau_i \succ \tau_j|\boldsymbol{\mu}) = \frac{1}{1+\exp(\boldsymbol{\mu}_j - \boldsymbol{\mu}_i)}$. Both models are well-established in the preference modelling literature and often behave similarly, but Thurstone's better matches the statistical assumptions of our method. The estimation of Σ also provides a mechanism for uncertainty-driven active learning.

[2]Notice how ε-bounding y_{ij} is required to ensure Φ^{-1} does not yield infinite values.

[3]See Appendix A for a brief discussion of the psychological assumptions underlying the feature expectation decomposition.

the state-action space into m disjoint subsets, which map to the m possible one-hot vectors. For trajectory τ_i, $\mathbf{n}_i$ can be interpreted as the number of timesteps spent in each subset, and the reward vector $\mathbf{r}$ as a set of *components* that reward state-action pairs according to the subsets they fall within. We model reward components as independent, so that C is a diagonal matrix. Furthermore, the partition induced by ϕ has a *binary tree* structure, with the m subsets as leaves, connected by a hierarchy of internal nodes emanating from a root. Each internal node applies a test to the state-action pair (s, a). If the test is passed, the logical flow proceeds to the "right" child node. Otherwise, it proceeds to the "left" child. Testing continues until a leaf node $x \in \{1..m\}$ is reached, the state-action pair is mapped to the corresponding one-hot vector, and the reward is given as the corresponding component $\mathbf{r}_x$ with variance C_{xx}. Although most of our algorithm does not rely on this assumption, we focus here on Euclidean state and action spaces $\mathcal{S} = \mathbb{R}^{D_s}$, $\mathcal{A} = \mathbb{R}^{D_a}$, in which a state-action pair is a vector $(s, a) \in \mathbb{R}^D$, where $D = D_s + D_a$. Internal node tests have the form $(s, a)_d \geq c : d \in \{1..D\}$, which evaluates whether the dth element of (s, a) meets or exceeds a threshold c. Consequently, each state-action subset has the geometry of an axis-aligned hyperrectangle. Figure 1 provides an illustrative example.

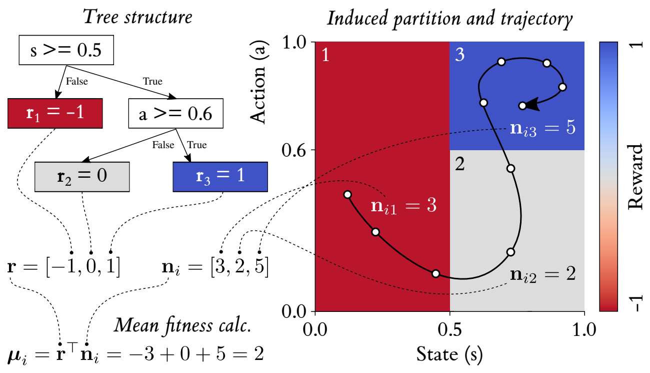

Figure 1: Tree and induced partition for a simple case with $D_s = D_a = 1$ and $m = 3$. For a trajectory τ_i, $\mathbf{n}_i$ counts the timesteps (white circles) spent in each subset, and its dot product with $\mathbf{r}$ gives the mean fitness estimate μ_i. Each component also has a variance C_{xx}, which can be used to compute the trajectory fitness variance $\Sigma_{ii} = n_i C n_i$ (not shown).

4 PROPOSED ALGORITHM

We now present an algorithm for optimising equation 5 subject to the tree-structuring constraint on ϕ, alongside an active learning scheme that adapts the sampling distribution ψ to focus on trajectories with high-variance fitness estimates and correct overestimation errors. An indirect, multi-stage optimisation approach was converged on after extensive experimentation with alternatives, which we outline in Appendix B. This approach is computationally efficient, easy to implement, and yields reward functions that are significantly more robust to small data changes than the alternatives. As the algorithm iterates over stages of preference elicitation, reward component fitting, tree structure refinement and distribution updates, there is no single first step. We have tried to order the following subsections to maximise comprehensibility.

4.1 Preference Elicitation and Representation

We first outline the process of sampling trajectory pairs and storing preference labels. We assume a finite data setting, in which the trajectory space $(\mathcal{S} \times \mathcal{A})^T$ is approximated by a sequence of n trajectories $\mathcal{T} = (\tau_1, ..., \tau_n)$, and the domain of ψ is restricted to $\mathcal{T}^2$. This distribution can thus be written as a matrix $\Psi \in [0, 1]^{n \times n}$: $\sum_{ij} \Psi_{ij} = 1, \Psi_{ii} = 0,\ \forall i \in \{1..n\}$. For the moment, let us take both $\mathcal{T}$ and Ψ as given; we discuss their origins in section 4.5. An instance of preference feedback is obtained by sampling a trajectory pair τ_i, τ_j with probability Ψ_{ij}, presenting the pair (e.g. by visualisation) to a human, and recording the resultant preference label $y_{ij} \in [\varepsilon, 1-\varepsilon]$. We represent a set of k feedback instances by three data structures, a set $\mathcal{P}$, a matrix $A \in \{-1, 0, 1\}^{k \times n}$ and a vector $\mathbf{y} \in \mathbb{R}^k$, which are incrementally assembled as preferences arrive. After sampling the kth pair τ_i, τ_j and observing y_{ij}, we add $\{\tau_i, \tau_j\}$ to $\mathcal{P}$, append a row to A in which the ith element is 1, the jth element is -1 and all other elements are 0, and append $\mathbf{y}_k = y_{ij}$. $\mathcal{P}$ thus serves as a record of which pairs have been sampled. A and $\mathbf{y}$ allow us to express equation 5 in matrix form as follows:

$$\underset{\phi, \mathbf{r}, C}{\text{argmin}} \left[\Phi^{-1}(\mathbf{y}) - (\text{diag}(NA^\top CAN^\top)^{-\frac{1}{2}})^\top AN^\top \mathbf{r}\right]^2. \qquad (7)$$

Here, N is the $m \times n$ matrix of columnar feature vectors $\mathbf{n}_i$. This notation is used in equations throughout this section.

4.2 Trajectory-Level Fitness Estimation

Direct optimisation of equation 7 subject to the tree-structuring constraint is computationally intractable, so we approximate the global problem by a sequence of local ones. First, we temporarily apply Thurstone's *Case V* reduction [27], which assumes unit standard deviations for all fitness differences, i.e. $\text{diag}(NA^\top CAN^\top)^{\frac{1}{2}} = \mathbb{1}_m$, so simplifies the bracketed expression in equation 7 to $\Phi^{-1}(\mathbf{y}) - AN^\top \mathbf{r}$. Following the method developed by Morrissey and Gulliksen [13], we then use least squares to compute a somewhat biased (by the Case V assumption) estimate of $N^\top \mathbf{r} = \boldsymbol{\mu}$, which is the $n \times 1$ vector of mean fitness estimates at the level of complete trajectories:

$$N^\top \mathbf{r} = \boldsymbol{\mu} \approx \tilde{\boldsymbol{\mu}} = (A^\top A)^{-1} A^\top \Phi^{-1}(\mathbf{y}). \qquad (8)$$

4.3 Independent Reward Component Fitting

We then use the trajectory-level estimates $\tilde{\boldsymbol{\mu}}$ to fit the means and variances of the reward components. This is a kind of temporal credit assignment problem: how does each component contribute to the overall fitness of a trajectory τ_i? To solve it, we recall that under our tree-structuring constraint, N_{xi} is the number of timesteps τ_i spends in the xth state-action subset. A priori, we cannot know which timesteps are responsible for the fitness of τ_i, so the least strong assumption is a uniform temporal prior, that the reward component for this subset contributes a fraction N_{xi}/T of the fitness. We can thereby compute an empirical estimate of the xth reward component mean by taking a weighted sum over trajectories:

$$\mathbf{r}_x = \frac{\sum_{\tau_i \in \mathcal{P}} \frac{N_{xi}}{T} \tilde{\boldsymbol{\mu}}_i}{\sum_{\tau_i \in \mathcal{P}} N_{xi}}. \qquad (9)$$

Note that this summation is only over trajectories for which at least one preference label has been provided (denoted, in a slight abuse of notation, by $\tau_i \in \mathcal{P}$). We then use $\tilde{\boldsymbol{\mu}}$ and $\mathbf{r}_x$ to empirically estimate the variance of the xth component, C_{xx}:

$$C_{xx} = \frac{\mathrm{RSS}(N_x)}{\sum_{\tau_i \in \mathcal{P}} N_{xi}}; \quad \mathrm{RSS}(N_x) = \sum_{\tau_i \in \mathcal{P}} N_{xi} \left[\frac{\tilde{\boldsymbol{\mu}}_i}{T} - \mathbf{r}_x \right]^2. \quad (10)$$

The residual sum of squares $\mathrm{RSS}(\cdot)$ is a useful intermediate function that we use again below. By this method, reward components are fitted independently, using only the corresponding rows of N. The independent treatment of components leaves C as a diagonal matrix.

4.4 Tree Structure Refinement

So far, we have assumed a fixed N matrix, and thus a fixed feature function ϕ, but this function can be modified by adding or removing internal nodes in the tree structure described above. Consider splitting the xth leaf node into two by replacing it with an internal node which tests whether the dth element of the (s, a) vector exceeds a threshold c. The effect of this split on the feature matrix N is to replace the xth row of N with a new $2 \times n$ matrix, denoted by $N^{[xdc]}$, representing how the number of timesteps each trajectory spends in the xth state-action subset is distributed between two new child subsets contained within it. Using the two rows of $N^{[xdc]}$, reward components for the child subsets can be fitted using equations 9 and 10. All other reward components remain unchanged. Repeatedly making such splits grows the tree, each time increasing the number of leaves, subsets and associated reward components m by 1.

Whenever the trajectory-level fitness estimates $\tilde{\boldsymbol{\mu}}$ are updated, our algorithm grows the existing tree by recursive splitting up to a maximum $m = m_{\max}$, at each step choosing a component to split x, splitting dimension d and threshold c to greedily minimise the total residual sum of squares $\sum_{x=1}^{m} \mathrm{RSS}(N_x)$, and thus achieve a better fit to $\tilde{\boldsymbol{\mu}}$. Since reward components are fitted independently, a single step of this optimisation process can be expressed as follows:

$$\max_{1 \le x \le m} \max_{1 \le d \le D} \max_{c \in C_d} \mathrm{RSS}(N_x) - \mathrm{RSS}(N_1^{[xdc]}) - \mathrm{RSS}(N_2^{[xdc]}). \quad (11)$$

C_d is a set of candidate split thresholds along dimension d. In our experiments, where dataset sizes are moderate, we exhaustively search over all values that occur in labelled trajectories: $C_d = \{(s, a)_d,\ \forall (s, a) \in \tau,\ \forall \tau \in \mathcal{P}\}$. Crucially, the RSS-based splitting criterion in equation 11 is precisely the one used in classical regression tree learning [6]. Our algorithm thus utilises a virtually-unmodified, highly-optimised, regression tree implementation.

Once $m_{\max}$ is reached, we then iterate backwards through the growth process, pruning the tree back until $m = 1$. At each step in this backward pass, we use the corresponding N, $\mathbf{r}$ and C to evaluate the labelling loss expression given in equation 7 (i.e. the global objective that we aim to optimise). To this labelling loss, we add a *complexity regularisation* term αm to modulate the tradeoff between predictive accuracy and interpretability (through compactness) and also to mitigate overfitting. We identify the tree size that minimises the regularised labelling loss, and use this tree as the updated ϕ.

4.5 Trajectory Pair Sampling Distribution

Our algorithm works in two data settings: offline and online. In the offline setting, the underlying trajectory dataset $\mathcal{T}$ remains fixed but the sampling matrix Ψ is modified over time. In the online setting, both Ψ and $\mathcal{T}$ change, with the latter being gradually augmented with new trajectories. This makes it possible to use trajectory data generated by the PbRL agent itself as it learns a policy in real-time.

Offline setting. A wide variety of active preference learning schemes have been proposed for PbRL [26]. Partly inspired by existing work in the bandit literature [7, 30] we adopt an upper confidence bound (UCB) strategy, which weights trajectory pairs according to optimistic estimates of their summed fitness. This strategy prioritises trajectories with highly uncertain fitness under the current model, for which additional preference labels are likely to be most useful for reducing uncertainty (in this respect, it is similar to [8]). Additionally, the optimism induces a bias towards identifying and correcting cases where trajectory fitness is overestimated, ultimately yielding a conservative reward function which counteracts the well-known overestimation bias in value-based RL [12]. Finally, biasing the preference dataset towards promising trajectories leads the reward function to prioritise distinguishing between high and very high fitness behaviour (rather than low vs very low), which reduces the risk of an agent stagnating at mediocre fitness with no incentive to improve. To implement the UCB strategy we use N, $\mathbf{r}$ and C to compute a vector of optimistic fitness values:

$$\mathbf{u} = \boldsymbol{\mu} + \lambda\ \mathrm{diag}(\Sigma)^{\frac{1}{2}} = N^\top \mathbf{r} + \lambda\ \mathrm{diag}(N^\top C N)^{\frac{1}{2}}, \quad (12)$$

where $\lambda \ge 0$ determines the number of standard deviations added to the mean. We then construct an $n \times n$ weighting matrix as follows:

$$W_{ij}^{\mathrm{off}} = \begin{cases} 0 & \text{if } i = j \text{ or } \{\tau_i, \tau_j\} \in \mathcal{P}, \\ & \text{or } (\mathcal{P} \ne \emptyset \text{ and } \tau_i \notin \mathcal{P}), \\ \mathbf{u}_i + \mathbf{u}_j + \delta & \text{otherwise,} \end{cases} \quad \begin{array}{l} \forall i \in \{1..n\}, \\ \forall j \in \{1..n\}. \end{array} \quad (13)$$

Here, the three "zeroing" conditions respectively prevent comparing a trajectory to itself, prevent repeated pairs, and ensure that one of any sampled pair has already received feedback.[4] The offset δ is calibrated so that the minimum element not matching a zeroing condition is set to 0.[5] If *all* elements match a zeroing condition, then all possible trajectory pairs have been sampled and the process must be halted. Otherwise, we define $\Psi = W^{\mathrm{off}} / \sum_{ij} W_{ij}^{\mathrm{off}}$.

Online setting. If $\mathcal{T}$ monotonically expands with new trajectories over time, but preference labels are obtained at a constant rate, it is possible to show that a higher density of labels is given to trajectory pairs that appear earlier, creating a strong earliness bias in the preference dataset. Assuming a total labelling budget $k_{\max}$ and known final trajectory count $n_{\max}$, we correct for this bias by collecting a batch of labels every time f_l new trajectories are added, using monotonically-increasing batch sizes. On the bth batch, we define W^{on} the same as W^{off}, with the extra condition that $W_{ij}^{\mathrm{on}} = 0$ if $i \le f_l(b-1)$ and $j \le f_l(b-1)$, which ensures that at least one of i and j are in the most recent f_l trajectories. We compute Ψ by normalising W^{on} as above, then sample k_b trajectory pairs, where

$$k_b = \mathrm{round}\left(k_{\max} \frac{f_l^2 (2b - 1) - f_l}{n_{\max}(n_{\max} - 1)} \right). \quad (14)$$

We refer the reader to Appendix C for a more detailed justification of this approach and a derivation of equation 14. Note that the offline setting is recovered by setting $f_l = n_{\max} = |\mathcal{T}|$.

[4] The last condition ensures that the graph representing the set of pairwise comparisons $\mathcal{P}$ is connected, meaning there is a path between any two $\tau_i \in \mathcal{P}$, $\tau_j \in \mathcal{P}$. As shown by Csató [9], this is necessary for the least squares solution in equation 8 to be unique.
[5] Unless this is also the maximum element, in which case it is offset to a positive value (nominally 1) to ensure that $\sum_{ij} W_{ij}^{\mathrm{off}} > 0$ and prevent a divide-by-zero error.

4.6 Complete Algorithm

Our algorithm initiates with one reward component ($m = 1$) and $\mathbf{r} = [0]$, $C = [0]$. During batch b, labelling is paused every f_u samples for an iteration of trajectory-level fitness estimation, reward component fitting and tree refinement. This leads to a modified $\mathbf{u}$ vector, which alters the sampling matrix for the rest of the batch. In the online setting, where an RL agent uses the learnt reward to train a policy in real-time, the latest ϕ and $\mathbf{r}$ are used as soon as an update is made. Otherwise, the algorithm runs until the budget k_{max} is expended, and the final fixed reward is used to train a policy at a later time. A pseudocode algorithm is given in Appendix D.

5 PERFORMANCE RESULTS

We have evaluated our algorithm in four RL environments under various learning conditions. In all cases we used a feedback budget k_{max} between 600 and 620, equating to $\approx$ 1 hour of human time. Full experimental details are reported in Appendix E; below we discuss our findings by reference to the blue annotation letters in figures 2-5. Key findings are in **bold** and summarised at the end.

5.1 Offline with Oracle Feedback (Figure 2)

Here, $\mathcal{T}$ was fixed, and comprised of trajectories generated by an RL agent (the *pilot agent*) as it trained on each environment's default, hand-engineered reward function. Preference labels came from a synthetic oracle with query access to this reward function. We then trained a second agent (the *PbRL agent*) on the learnt reward, and finally measured the PbRL agent's alignment to the original reward function, which was taken to represent ground-truth fitness; see [16] for a similar method of quantitative evaluation. We show learning curves (time series of fitness per episode) for both learnt (a) and ground-truth (b) reward (5 repeats; mean and min-max range shown). The consistent monotonicity of the former indicates that tree-structured reward functions gave rise to stable agent learning in all cases, and the latter confirm that this learning was well-aligned with the ground-truth. **For Pendulum and RoboCar, asymptotic fitness was indistinguishable from the pilot, while for LunarLander and FoodLava, it was slightly below**. We show additional plots for FoodLava. (c) shows that the αm-regularised labelling loss was minimised by a tree with 9 leaves, hence 9 reward components. (d) gives another measure of alignment by plotting true vs learnt fitness ($\pm$1 std) for both the pilot trajectories $\mathcal{T}$ and those generated during PbRL agent training. In both cases there is a clear positive correlation, although the relationship for the latter is noisier, suggesting a degree of distributional shift. (e) visualises the 9 reward components over the two state dimensions as coloured rectangles.[6] The components are arranged isomorphically to the maze layout, with negative reward in the red "lava" region and positive reward around the green "food". However, some misalignment is visible as high-reward component 8 is too large in the vertical direction. Referring to (f), which plots the final 10 trajectories of all 5 PbRL repeats, we see that this misalignment led to policies that sometimes terminated just below the food.

[6]This and subsequent visualisations use a projection method introduced in [4], which represents the state-action subset for each component by its rectangular projection onto two dimensions, coloured by its mean reward. Where the projections of multiple subsets overlap, their colours are averaged, weighted by the number of samples falling within them in the trajectory set $\mathcal{T}$. In this first FoodLava visualisation, there are in fact no overlaps because the tree only contains splits along the two plotted dimensions.

5.2 Offline with Human Feedback (Figure 3)

Using the same pilot trajectories as $\mathcal{T}$, we then gathered preference data from 62 human participants via a survey (a) then trained PbRL agents using the resultant reward functions. We asked participants to indicate their level of relevant expertise (b) and expectations of the likelihood of our method succeeding, both before and after completing the survey (c), and to rank the tasks by perceived difficulty of giving feedback (d). Notably, this ranking turned out to predict the success of our method in this setting, since **for LunarLander and Pendulum we achieved asymptotic ground-truth fitness (e) only slightly below the oracle results**. Information about the ground-truth was not used anywhere in this experiment, which instead relied on participants' intuitive understanding of the tasks. The fact that it could nonetheless be well-optimised by the resultant PbRL agents indicates that for LunarLander and Pendulum, **human intuition was broadly aligned with the default reward functions**. Again, the coloured rectangle plots (f) and (g) provide insight into the learnt reward structure, with high reward in LunarLander given in a column above the landing zone, and in Pendulum given when the pole is upright with small angular velocity. The final 10 trajectories from the PbRL runs are overlaid, showing that the agents sought out high-reward regions and consequently solved the respective landing and pole-balancing tasks. **We were unable to achieve aligned learning in FoodLava and RoboCar**, although in the former the outcome was not as catastrophic as the learning curve suggests. As can be seen in the rectangle/trajectory plot (h), the PbRL agent learnt to solve most of the maze but was not incentivised to proceed to the food, since a large positive reward component covered the entire upper third. This, we hypothesise, is evidence of a **causal confusion problem**: within the pilot run dataset, most trajectories that reached the upper third then went on to the food, so our uniform temporal credit assignment was unable to determine that the latter step was necessary for a favourable preference label. A similar issue arose in RoboCar, which we focus on in our interpretability analysis (section 6). For Pendulum, we include the labelling loss curve (i) and alignment plot (j), alongside box plots showing the agreement between provided preference labels and ground-truth fitness differences (k). Ratings from participants of all expertise levels generally aligned with fitness (above dotted line), with this trend becoming slightly more pronounced for more certain ratings (closer to 10), and an indication that those with RL expertise exhibited somewhat lower variance than those without.

5.3 Online with Oracle Feedback (Figure 4)

Next, we deployed our algorithm in an online setting, using trajectories generated by the PbRL agent as it trained on the learnt reward in real-time. After n_{max} trajectories were gathered, the reward function was fixed and the agent continued to train until convergence. Initially, we used synthetic oracle feedback. The ground-truth learning curves (a) indicate that overall performance was similar to the oracle-based offline setting, **with mean final fitness being slightly higher for Pendulum and FoodLava and slightly lower for LunarLander and RoboCar**. We further examine one of the five RoboCar runs via a hybrid visualisation that we call a *learning timeline* (b). With $n_{\text{max}} = 200$ and $f_l = 10$ we had a total of 20 labelling batches, over which the number of labels k increased according to the scheduling equation 14. The heatmap

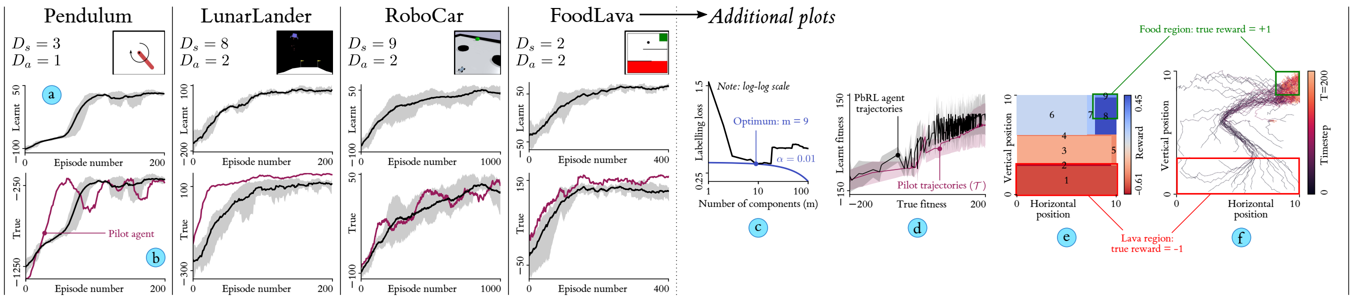

Figure 2: Performance in offline setting using oracle feedback; additional plots shown for FoodLava.

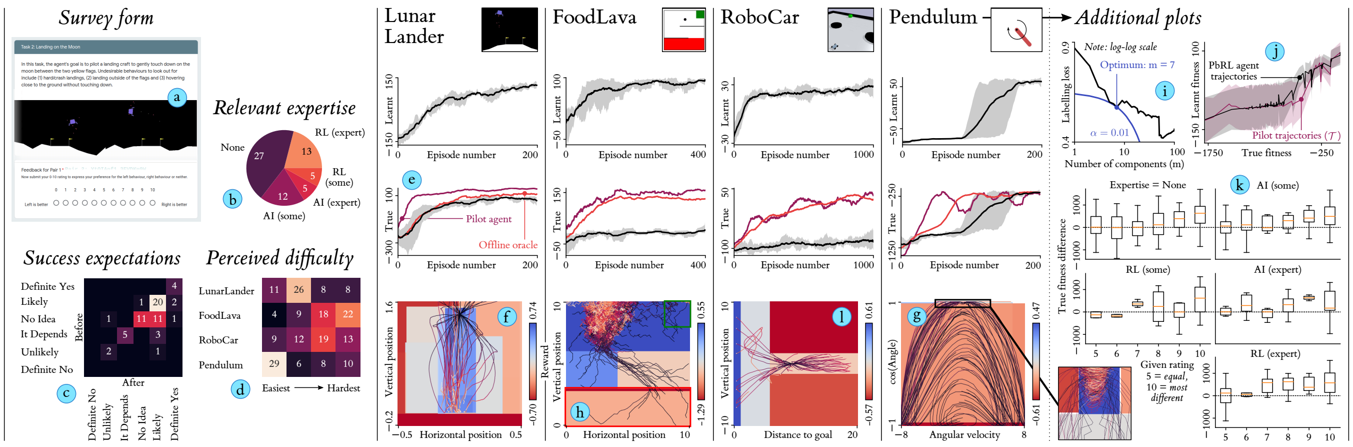

Figure 3: Performance in offline setting using human feedback; additional plots shown for Pendulum.

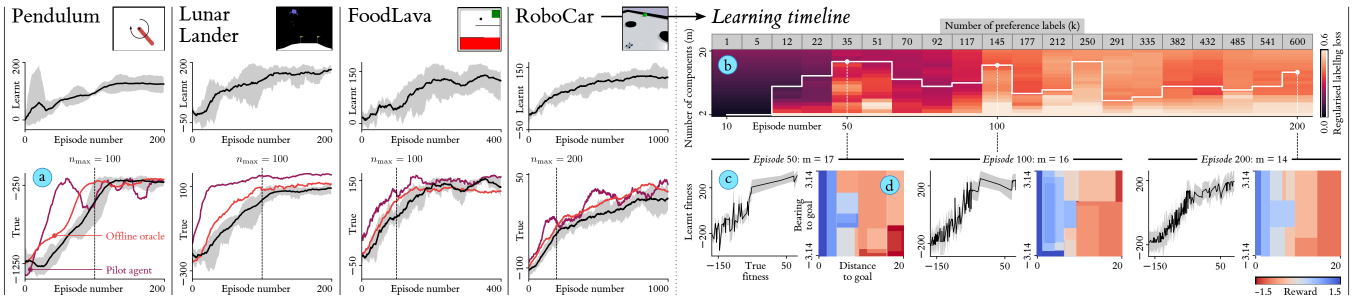

Figure 4: Performance in online setting using oracle feedback; learning timeline shown for RoboCar.

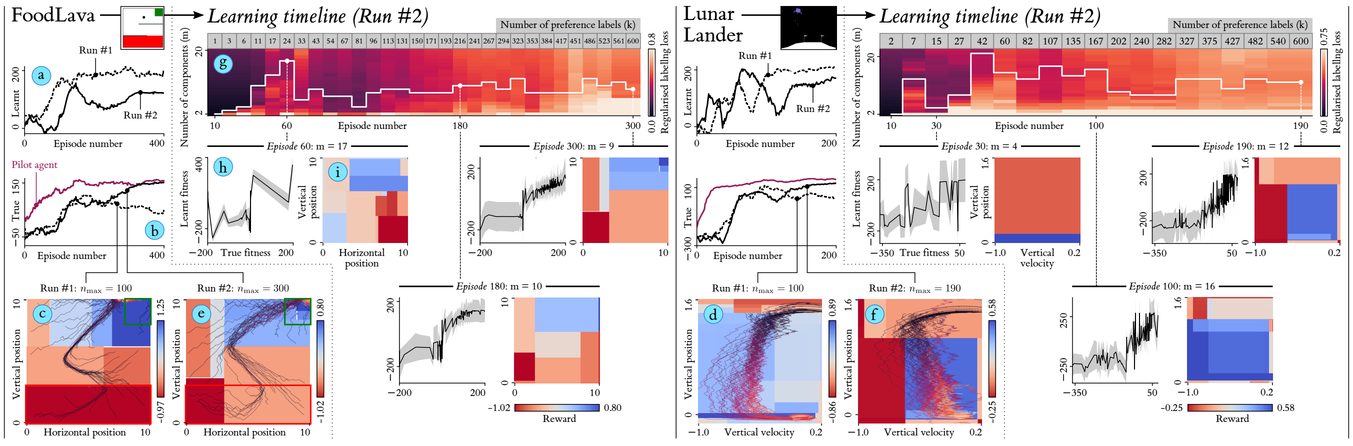

Figure 5: Performance in online setting using human feedback; results with learning timelines for FoodLava and LunarLander.

shows how the regularised labelling loss varied as a function m during the tree pruning sweep performed after each batch. The overlaid white curve shows how the tree size was modified accordingly to track the optimal m. As batches accumulated, the global pattern was that m **first increased to a maximum, then remained somewhat below that maximum thereafter**, with large changes becoming less frequent. Inspecting the model at three checkpoints during training, we find that (c) **the positive correlation between true and learnt fitness became less noisy over time**, and (d) the reward components converged to an arrangement that positively rewarded both proximity to the goal and facing towards it (bearing ≈ 0), doing so in an almost-symmetric manner.

5.4 Online with Human Feedback (Figure 5)

Finally, we ran the algorithm online using feedback from a single human participant. Due to the labour-intensiveness of this experiment, we focused on two environments – FoodLava and LunarLander – both of which revealed the risk of prematurely fixing the reward structure. For both, we initially used $n_{\max} = 100$ as in the oracle-based experiments, and found the agent quickly converged to high fitness according to the learnt reward function (a; dotted lines) but after episode $n_{\max}$, gradually lost fitness according to the ground-truth reward (b). This indicated that **the reward structure had been fixed in a state that was only partly aligned**, so that further optimisation hindered true performance. Using rectangle plots of the final reward components, we can see that for FoodLava (c), the maximum positive reward component was placed around the food region, but was "loosely" targeted as it exceeded the bounds of the food. The overlaid final trajectories show that the agent learnt to seek out this high reward, but sometimes stop short of the food itself. For LunarLander (d), a similarly "loose" reward function was learnt that gave high reward for a vertical position close to zero, regardless of the vertical velocity. The trajectory overlay indicates this reward function led the agent to maintain high negative velocity as it approached the ground, which was recognised as a crash landing by the ground-truth reward function. For both environments, we completed a second run using a higher $n_{\max}$, thereby **distributing the same feedback budget over a larger fraction of the training process**, providing more time for the human to reactively fine-tune alignment. For these runs, we found that ground-truth fitness (b) **almost matched the pilot agent** from the offline oracle experiment, with **no sign of a performance dropoff**. For FoodLava, the rectangle plot of reward components (e) reveals a much smaller area of maximum reward that did not exceed the bounds of the food region and thus incentivised the agent to reliably enter it. For LunarLander (f), we had a very different reward structure to the first run, which gave positive reward for maintaining slow vertical velocity, and negative reward for exceeding a velocity threshold of -0.55, rather than merely rewarding reaching the ground. This incentivised the agent to gradually decelerate as its height decreased, resulting in softer landings that were not registered as crashes by the ground-truth reward function. For both environments, we show a learning timeline for the second run (g). We again see the trend of m increasing to a maximum before stabilising at a lower value. The checkpoints also show the relationship between true and learnt fitness becoming less noisy over time (h) and several intermediate reward component layouts that emerged during training (i).

Summary of Key Findings. • With no exceptions, the tree-structured reward functions gave rise to stable, convergent reinforcement learning. • Using several hundred instances of oracle feedback based on ground-truth reward functions, our algorithm could reconstruct those reward functions sufficiently well to train PbRL agents whose performance nearly matched that of conventional RL agents. • The aggregated preferences of 62 human participants (offline), as well those of a single participant (online), yielded learnt reward functions that were similarly well-aligned to the ground-truth. This is despite participants having no direct knowledge of the ground-truth, instead relying on intuitive task understanding alone. • In the offline setting, dataset biases led to causal confusion, where the learnt reward incentivised state-action pairs that commonly appeared *alongside* high-fitness behaviours, as well as the behaviours themselves. Careful rebalancing of training data, or moving to an online learning setup, would both help to mitigate this problem. • In the online setting, the main failure mode was "loose" alignment due to fixing the reward prematurely. Increasing $n_{\max}$ gave more time to reactively correct for misaligned changes in behaviour. • In the online setting, the number of components m tended to initially increase rapidly, then stabilise somewhat below the maximum later on, with large changes becoming less frequent. • In all human experiments, we did not encounter a single tree with a split along an action dimension, so that all rewards were a function of state only. We are wary to make a general claim about this result, but it is consistent with a recent suggestion that human teaching focuses on states over actions [17], and indicates future PbRL work may succeed by learning state-dependent rewards only.

6 INTERPRETABILITY DEMONSTRATION

The analysis of rectangle plots above, which provides insight into the learnt reward functions and their effect on behaviour, exemplifies the interpretability benefits of the tree-structuring constraint. Figure 6 demonstrates these benefits further by focusing on two specific PbRL runs: one failure case and one success case.

Failure case: RoboCar using offline human feedback. As is visible in figure 3 (l), this reward function led to policies that sometimes reached the goal as desired, but other times made no move towards the goal and appeared to seek only to maintain a vertical position close to 0. We can diagnose this misalignment by examining the full reward function tree (figure 6; a). Here, the splitting dimensions are y = vertical position (initialised to 0), d = distance to goal, and β = bearing to goal in radians. The first two splits appear well-aligned, creating a component with maximum reward for achieving $d < 1.16$ and a smaller positive reward for $d < 5.84$. The remaining splits are problematic, creating components that penalise moving out of the region $y \in [-1.64, 1.68]$ and, otherwise, reward a bearing outside of $\beta \in [-2.15, 2.18]$ (i.e. facing *away* from the goal). To understand this, consider the design of the environment. In each episode, the goal position is randomised but the car is initialised facing to the right, making it easier to reach the goal when it is also to the right. Hence, a majority of goal-reaching trajectories in the pilot run dataset showed the car driving directly forward, rarely exiting a narrow corridor around $y = 0$. The splits to penalise large absolute y are thus an example of causal confusion, in which behaviour correlating with a high fitness outcome is mistaken for being high fitness in itself, and would not appear if the environment were differently initialised or the dataset better balanced. We give a similar, if subtler, causal confusion justification of the β-based splits in Appendix F. The heatmaps (b), (c) and (d) provide fine-grained insight into the effect of the misaligned reward function on the learning dynamics of one of the 5 PbRL repeats. (b) represents the timesteps spent in each component – the N matrix – over the 1000-episode training history. Multiplying N row-wise with the mean reward vector $\mathbf{r}$ we obtain (c), which gives per-episode reward from each component, and can be understood as a *decomposed learning curve*. Summing (c) column-wise gives (d), the total fitness for each episode, which is a conventional learning curve. From these, we find that the agent quickly (by episode 50) learned to avoid negative-reward components 3 and 8, inducing an early bias towards exiting the $y \in [-1.64, 1.68]$ corridor. With this bias in place, exploration was curtailed and the agent largely settled into the moderate positive rewards of components 4 and 7. Although there was a gradual increase in visitation to component 1 (the one corresponding to reaching the goal) in the first half of training, the agent never completely prioritised this component, with visitation peaking around episode 700 before dropping off again. In (e), we harness the component structure to construct textual *report cards* for two episodes near the end of training (950 and 975) which describe the state-action subsets that were visited. While both are in the top 10% of episodes by performance on the learnt reward, the former is aligned (obtaining positive reward from components 1 and 2) while the latter is not (staying entirely in component 6, thereby driving straight ahead despite the goal being behind it).

Success case: FoodLava using online oracle feedback. In this run (chosen randomly from the 5 repeats), we achieved aligned learning. For the first $n_{\text{max}} = 100$ episodes, a label batch was obtained at intervals of $f_l = 10$ and the tree structure incrementally updated by growth and pruning. (f) depicts the net changes resulting from each update using both rectangle plots and a graph of the split/merge dependencies between components from batch-to-batch. Key events in the construction of the reward function included the merging of four components into one at $b = 4$, and the corrective splitting, merging, and re-splitting (at a different threshold) of a component between $b = 8$ and $b = 10$, yielding the final maximum-reward component 9 whose subset boundaries (horizontal position $x \geq 7.95$, vertical position $y \geq 8.06$) lined up almost exactly with the food region ($x \geq 8$, $y \geq 8$). From $b = 2$ onwards, component 1 was persistent, being neither split nor merged. However, its mean and variance were continually refined as preference labels arrived, a process that we visualise in (g). For each batch, every trajectory τ_i that had been labelled so far (of which there are more for later batches) is shown as a black horizontal line, whose vertical position corresponds to its trajectory-level fitness estimate $\tilde{\boldsymbol{\mu}}_i$ (divided by the episode length $T = 200$) and whose width is proportional to the time spent in component 1, N_{1i}. The method described in section 4.3 effectively fits a normal distribution to these lines, and the results for all batches are overlaid (mean as squares, ±1 std as shading). Between batches 2 and 10, component 1 saw a slight increase in its mean, and a gradual narrowing of its variance, as more trajectories were labelled. (h) depicts the fixed 9-component tree used for the final $n_{\text{post fix}} = 300$ training episodes, and (i) shows the timesteps spent in each component throughout this period. As of episode 100, roughly equal time was being spent in components 2 and 9, but the latter came to dominate around episode 250. Since component 9 corresponded to the food region, this indicated consistent, rapid solution of the navigation problem, with occasional failures (j) attributable to the agent becoming stuck in component 1.

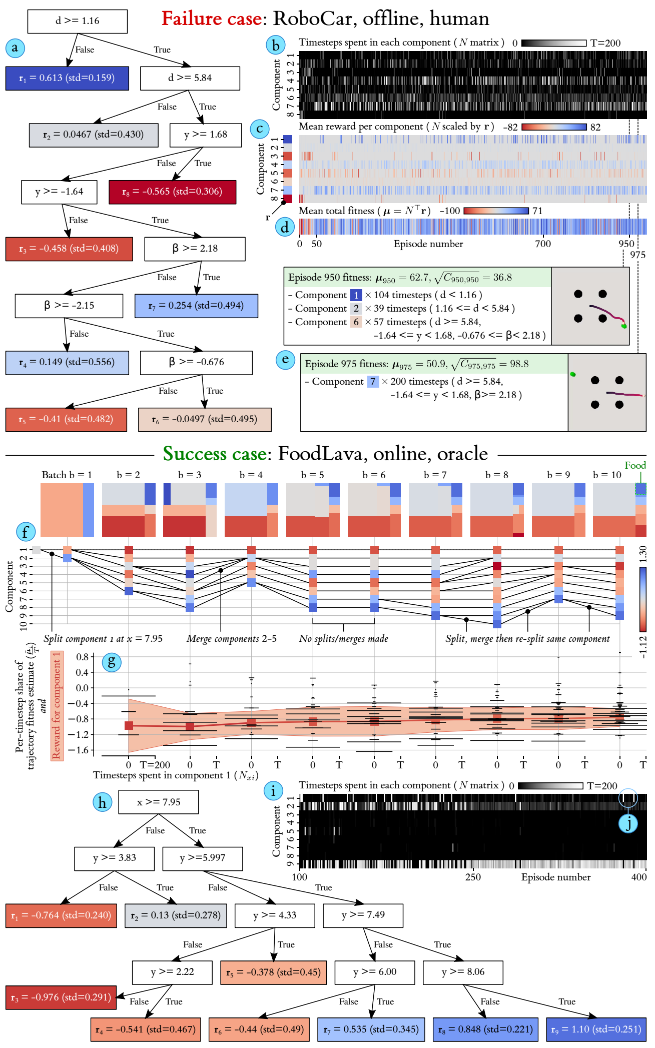

Figure 6: Interpretability demo for two exemplar cases.

7 CONCLUSION

We have presented an algorithm for interpretable PbRL using tree-structured reward functions, demonstrating successful learning of compact and aligned reward functions across four environments, alongside informative and actionable failure cases due to causal confusion in the offline setting and premature reward fixing in the online setting. We have also shown the value of interpretability for exploring and debugging the learnt reward structure. In the offline setting, future work could construct trajectory datasets using unsupervised agents that optimise for behavioural diversity (e.g. [11]) instead of our somewhat contrived pilot agents. In the online setting, there is scope for larger human experiments, with a focus on ablation and hyperparameter tuning. An additional layer of interpretability could be realised by integrating our method with agent architectures that learn decomposed value functions [18].

REFERENCES

[1] Pieter Abbeel and Andrew Y Ng. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*. 1.

[2] Riad Akrour, Marc Schoenauer, and Michèle Sebag. 2012. April: Active preference learning-based reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 116–131.

[3] Christian Arzate Cruz and Takeo Igarashi. 2020. A survey on interactive reinforcement learning: Design principles and open challenges. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*. 1195–1209.

[4] Tom Bewley and Jonathan Lawry. 2021. TripleTree: A Versatile Interpretable Representation of Black Box Agents and their Environments. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11415–11422.

[5] Ralph Allan Bradley and Milton E Terry. 1952. Rank analysis of incomplete block designs: I. The method of paired comparisons. *Biometrika* 39, 3/4 (1952), 324–345.

[6] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. 2017. *Classification and regression trees*. Routledge.

[7] Alexandra Carpentier, Alessandro Lazaric, Mohammad Ghavamzadeh, Rémi Munos, and Peter Auer. 2011. Upper-confidence-bound algorithms for active learning in multi-armed bandits. In *International Conference on Algorithmic Learning Theory*. Springer, 189–203.

[8] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep Reinforcement Learning from Human Preferences. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30.

[9] László Csató. 2015. A graph interpretation of the least squares ranking method. *Social Choice and Welfare* 44, 1 (2015), 51–69.

[10] Daniel Dewey. 2014. Reinforcement learning and the reward engineering principle. In *2014 AAAI Spring Symposium Series*.

[11] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. 2018. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070* (2018).

[12] Scott Fujimoto, Herke Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*. PMLR, 1587–1596.

[13] Harold Gulliksen. 1956. A least squares solution for paired comparisons with incomplete data. *Psychometrika* 21, 2 (1956), 125–134.

[14] Yuan Guo, Peng Tian, Jayashree Kalpathy-Cramer, Susan Ostmo, J Peter Campbell, Michael F Chiang, Deniz Erdogmus, Jennifer G Dy, and Stratis Ioannidis. 2018. Experimental Design under the Bradley-Terry Model.. In *IJCAI*. 2198–2204.

[15] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. 2017. Inverse Reward Design. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/32fdab6559cdfa4f167f8c31b9199643-Paper.pdf

[16] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. 2018. Reward learning from human preferences and demonstrations in Atari. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31.

[17] Snehal Jauhri, Carlos Celemin, and Jens Kober. 2020. Interactive Imitation Learning in State-Space. In *Conference on Robot Learning*.

[18] Zoe Juozapaitis, Anurag Koul, Alan Fern, Martin Erwig, and Finale Doshi-Velez. 2019. Explainable reinforcement learning via reward decomposition. In *IJCAI/ECAI Workshop on Explainable Artificial Intelligence*.

[19] Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. 2018. Scalable agent alignment via reward modeling: a research direction. *arXiv preprint arXiv:1811.07871* (2018).

[20] Richard Maclin and Jude W Shavlik. 1996. Creating advice-taking reinforcement learners. *Machine Learning* 22, 1 (1996), 251–281.

[21] Eric J Michaud, Adam Gleave, and Stuart Russell. 2020. Understanding learned reward functions. *arXiv preprint arXiv:2012.05862* (2020).

[22] Frederick Mosteller. 1951. Remarks on the method of paired comparisons: I. The least squares solution assuming equal standard deviations and equal correlations. *Psychometrika* 16, 1 (1951), 3–9.

[23] Jacob Russell and Eugene Santos. 2019. Explaining reward functions in markov decision processes. In *The Thirty-Second International Flairs Conference*.

[24] David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. 2021. Reward is enough. *Artificial Intelligence* (2021), 103535.

[25] Satinder Singh, Richard L Lewis, and Andrew G Barto. 2009. Where do rewards come from?. In *Proceedings of the annual conference of the cognitive science society*. Cognitive Science Society, 2601–2606.

[26] Aditi Talati, Dorsa Sadigh, et al. 2021. APReL: A Library for Active Preference-based Reward Learning Algorithms. *arXiv preprint arXiv:2108.07259* (2021).

[27] Louis L Thurstone. 1927. A law of comparative judgment. *Psychological review* 34, 4 (1927), 273.

[28] Christian Wirth, Riad Akrour, Gerhard Neumann, Johannes Fürnkranz, et al. 2017. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research* 18, 136 (2017), 1–46.

[29] Yichong Xu, Ruosong Wang, Lin Yang, Aarti Singh, and Artur Dubrawski. 2020. Preference-based Reinforcement Learning with Finite-Time Guarantees. *Advances in Neural Information Processing Systems* 33 (2020).

[30] Masrour Zoghi, Shimon Whiteson, Remi Munos, and Maarten Rijke. 2014. Relative upper confidence bound for the k-armed dueling bandit problem. In *International conference on machine learning*. PMLR, 10–18.