Backward explanation via redefinition of predicates

Léo Saulières **Martin C. Cooper** **Florence Dupin de Saint-Cyr**
IRIT, University of Toulouse III, France
{first name}.{last name}@irit.fr

Résumé

Les 'History eXplanations based on Predicates' (HXPs) étudient les historiques de comportement d'un agent ayant appris par renforcement à travers le prisme de prédicats [21]. Pour ce faire, un score d'importance est calculé pour chaque action de l'historique. Les actions les plus importantes sont alors affichées à l'utilisateur. Le calcul de score d'importance étant difficile (#W[1]-dur), il est nécessaire pour des historiques longs d'approximer les scores, au détriment de leur qualité. Aussi, nous proposons une autre méthode intitulée 'Backward-HXP' afin de fournir des explications pour ces historiques, et ce sans avoir à approximer les scores. Les expérimentations confirment la pertinence des 'Backward-HXP'.

Abstract

History eXplanation based on Predicates (HXP), studies the behavior of a Reinforcement Learning (RL) agent in a sequence of agent's interactions with the environment (a history), through the prism of an arbitrary predicate [21]. To this end, an action importance score is computed for each action in the history. The explanation consists in displaying the most important actions to the user. As the calculation of an action's importance is #W[1]-hard, it is necessary for long histories to approximate the scores, at the expense of their quality. We therefore propose a new HXP method, called Backward-HXP, to provide explanations for these histories without having to approximate scores. Experiments confirm the usefulness of Backward-HXP.

1 Introduction

Nowadays, Artificial Intelligence (AI) models are used in a wide range of tasks in different fields, such as medicine, agriculture and education [12, 8, 4]. Most of these models cannot be explained or interpreted without specific tools, mainly due to the use of neural networks which are effectively black-box functions. Numerous institutions [17, 9] and researchers [7, 15] have emphasized the importance of providing comprehensible models to end users. This is why the eXplainable AI (XAI) research field, which consists in providing methods to explain AI behavior, is flourishing. In this context, we propose a method for explaining AI models that have learned using Reinforcement Learning (RL).

In RL, the agent learns by trial and error to perform a task in an environment. At each time step, the agent chooses an action from a state, arrives in a new state and receives a reward. The dynamics of the environment are defined by the non-deterministic transition function and the reward function. The agent learns a policy π to maximize its reward; this policy assigns an action to each state (defining a deterministic policy). Our eXplainable Reinforcement Learning (XRL) method is restricted to the explanation of deterministic policies.

Various works focus on explaining RL agents using a notion of importance. To provide a visual summary of the agent's policy, Amir and Amir [2] select a set of interactions of the agent with the environment

(sequences) using the "state importance" [5]. From a set of sequences, Sequeira et Gervasio propose to learn a set of information, to deduce interesting elements to show the user in the form of a visual summary [22]. Using a self-explainable model, Guo et al. determine the critical time-steps of a sequence for obtaining the agent's final reward [11].

To explain an RL agent, explanation must capture concepts of RL [16]. To this end, the HXP method [21], consists of studying a history of agent interactions with the environment through the prism of a certain predicate. A predicate d can represent any partial description of states. This XRL method answers the question: *"Which actions were important to ensure that d was achieved, given the agent's policy π?"*. This paper continues the work on this method, by proposing a new way of defining the important actions of a history, called Backward-HXP (B-HXP). Specially, B-HXP was investigated because of the limits of (forward) HXP in explaining long histories.

The paper is structured as follows. The theoretical principle of HXP is outlined in Section 2, before defining B-HXP in Section 3. Section 4 presents the experimental results carried out on 3 problems. Section 5 presents related works and Section 6 concludes.

2 HXP

An RL problem is modeled using a Markov Decision Process [23], which is a tuple $\langle \mathcal{S}, \mathcal{A}, R, p \rangle$. $\mathcal{S}$ represents the state space and $\mathcal{A}$ the action space. $A(s)$ denotes the set of available actions to perform from s. $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and $p : \mathcal{S} \times \mathcal{A} \to Pr(\mathcal{S})$ are respectively the reward function and the transition function of the environment. $p(s'|s,a)$ represents the probability of reaching state s', having performed action a from state s. $\pi : \mathcal{S} \to \mathcal{A}$ denotes a deterministic policy that maps an action a to each state s; thus, $\pi(s)$ is the action performed by the agent in state s. Starting by doing an action a from a state s, the probability of a state s' is the product of the probabilities along the current path reaching s' according to π and p. In the following, we use the function *next* to compute the next possible states (associated with their probabilities) given a set of (state, probability) pairs S: $next_{\pi,p}(S) = \{(s', pr \times p(s'|s,a))$ such that $(s, pr) \in S$, $a = \pi(s)$ and $p(s'|s,a) \neq 0\}$. In order to compute the set of final states reachable at horizon k using the agent's policy π from a set of states S, the function $succ^k_{\pi,p}$ is defined recursively by $succ^0_{\pi,p}(S) = S$ and $succ^{n+1}_{\pi,p}(S) = next_{\pi,p}(succ^n_{\pi,p}(S))$.

HXPs provide to the user important actions for the respect of a predicate d, given an agent's policy π, by computing an importance score for each action in the history [21]. The language used for the predicate is based on the features that characterize a state. We consider a set of features $\mathcal{F} = \{f_1, ..., f_n\}$, where each feature f_i has a range of values defined by a domain D_i. The feature space is therefore $\mathbb{F} = D_1 \times ... \times D_n$. The state space $\mathcal{S}$ is a subset of $\mathbb{F}$. A predicate is given by a propositional formula with literals of the form $l_{i,j}$, where $l_{i,j}$ means that the feature f_i takes the value j in domain D_i. The importance score represents the benefit of performing an action a from s rather than another action $a' \in A(s) \backslash \{a\}$, where this benefit is the probability of reaching a state at a horizon of k that satisfies d. To evaluate an action we first require the notion of utility of a set of (state, probability) pairs.

Definition 1 (utility)**.** Given a predicate d, *the utility u_d of a set of (state, probability) pairs S* is:

$$u_d(S) = \sum_{(s,pr) \in S, s \models d} pr$$

Finally, the importance of an action a from a state s is defined as follows.

Definition 2 (importance)**.** Given a predicate d, an agent's policy π, a transition function p, the *importance score of a* from s at horizon k is defined by:

$$imp^k_{d,\pi,p}(s,a) = u_d(succ^k_{\pi,p}(S_{(s,a)})) - \operatorname*{avg}_{a' \in A(s) \backslash \{a\}} u_d(succ^k_{\pi,p}(S_{(s,a')})) \quad (1)$$

where avg is the average and $S_{(s,a)}$ is the set of reachable states (along with their probabilities) from s by performing action a. Formally, we have:

$$S_{(s,a)} = \{(s', p(s'|s,a)) \mid p(s'|s,a) \neq 0\}$$

The importance score lies in the range $[-1, 1]$, where a positive (negative) score denotes an important (resp. not important) action in comparison with other possible actions. Its computation is $\#W[1]$-hard [21], so it is necessary to approximate it, in particular by generating only part of the length-k scenarios with the *succ* function.

Approximate HXP consists in considering the last n time steps as deterministic over a horizon k, taking only the most probable transition into account. Thus, with b denoting the maximum number of transitions from a state-action couple (s, a) of a given RL problem, we produce at most b^{k-n} scenarios. As a note, the scenarios generated are not necessarily the most probable ones, since taking the most probable transition at each time step does not ensure that the most probable sequence is obtained.

To handle long histories on problems where the number of possible transitions is large, i.e. k and b large, it is necessary to use approximate methods to provide explanations in reasonable time, at the expense of only approximating the importance scores. In the next section, we propose a new way of computing HXP in a step-by-step backward approach, which allows us to provide explanations in reasonable time for long histories, without having to approximate the scores calculation. As we will see, this leads to other computational difficulties. The result is thus a novel method for the explanation of histories with different pros and cons compared to forward-based HXP.

3 Backward-HXP (B-HXP)

The idea of B-HXP is to iteratively look for the most important action in the near past of the state that respects the predicate under study. When an important action is found, we look at its associated state to define the new predicate to be studied. Indeed, by observing only a subset of the actions in the history (near past), the horizon for calculating importance scores is relatively small. In this sense, importance scores can be calculated exhaustively. The predicate is then modified so that actions can be evaluated with respect to a predicate that they can achieve within a shorter horizon. The following example will be used throughout this section to illustrate the method.

Example 1. *Consider the end of Bob's day. The history of Bob's actions is: [work, shop, watch TV, nap, eat, water the plants, read]. Bob's state is represented by* 5 *binary features: hungry, happy, tired, fridge, fuel. Fridge and fuel means respectively that the fridge is full and that the car's fuel level is full. Bob's last state is: (¬hungry, happy, tired, ¬fridge, ¬fuel) (for the sake of conciseness, Bob's states are represented by a boolean 5-tuple. Thus, Bob's last state is:* $(0, 1, 1, 0, 0)$*) The environment is deterministic and the predicate under study is "Bob is not hungry". We are looking for the* 2 *most important actions for Bob not to be hungry. Starting from the final state, the most important action in the near past is 'eat'. We are interested in its associated state, i.e. the state before doing the action 'eat', which is assumed to be* $(1, 0, 0, 1, 0)$*. The new predicate deduced from this state is "Bob is hungry and has a full fridge". In the near past of* $(1, 0, 0, 1, 0)$*, the 'shop' action is the most important one (among work, shop, watch TV and nap) for respecting this new predicate. To sum up, we can say that the reason that Bob is not hungry in the final state is that he went shopping (to fill his fridge) and then ate.*

Before describing in detail the B-HXP method, we introduce some notation. $H = (s_0, a_0, s_1, ..., a_{k-1}, s_k)$ denotes a length-k history, with $H_i = (s_i, a_i)$ denoting the state and action performed at time i, and for $i < j$, $H_{(i,j)}$ denotes the sub-sequence $H_{(i,j)} = (s_i, a_i, ..., s_j)$. In this section, we employ the term 'utility of a state s' to express the utility of the agent's action associated with s (this action being unique since we assume a deterministic policy). To define the near past of a state in H, it is necessary to introduce the maximum length of sub-sequences: l. This length must be sufficiently short to allow importance scores to be calculated in a reasonable time. The value of l depends on the RL problem being addressed, and specifically on the maximal number of possible transitions from any observable state-action pair,

namely b. It follows that the lower b is, the higher l can be chosen to be.

To provide explanations for long histories, we need a way of defining new intermediate predicates (such as Bob is hungry and the fridge is full in Example 1). For this we use Probabilistic Abductive eXplanations, shortened to PAXp [13].The aim of this formal explanation method is to explain the prediction of a class c by a classifier κ by providing an important set of features among $\mathcal{F}$. Setting these features guarantees (with at probability at least δ) that the classifier outputs class c, whatever the value of the other features. A classifier maps the feature space into the set of classes: $\kappa : \mathbb{F} \rightarrow \mathcal{K}$. We represent by $\mathbf{x} = (x_1, ..., x_n)$ an arbitrary point of the feature space and $\mathbf{v} = (v_1, ..., v_n)$ a specific point, where each v_i has a fixed value of domain D_i. [13] defines a weak PAXp as a subset of features for which the probability of predicting the class $c = \kappa(\mathbf{v})$ is above a given threshold δ when these features are fixed to the values in $\mathbf{v}$. A PAXp is simply a subset-minimal weak PAXp.

Definition 3 (PAXp [13])**.** Given a threshold $\delta \in [0, 1]$, a specific point $\mathbf{v} \in \mathbb{F}$ and the class $c \in \mathcal{K}$ such that $\kappa(\mathbf{v}) = c$, $\mathcal{X} \subseteq \mathcal{F}$ *is a weak PAXp* if:

$$Prop(\kappa(\mathbf{x}) = c \mid \mathbf{x}_{\mathcal{X}} = \mathbf{v}_{\mathcal{X}}) \geq \delta$$

where $\mathbf{x}_{\mathcal{X}}$ and $\mathbf{v}_{\mathcal{X}}$ are the projection of $\mathbf{x}$ and $\mathbf{v}$ onto features $\mathcal{X}$ respectively and $Prop(\kappa(\mathbf{x}) = c \mid \mathbf{x}_{\mathcal{X}} = \mathbf{v}_{\mathcal{X}})$ is the proportion of the states $\mathbf{x} \in \mathbb{F}$ satisfying $\mathbf{x}_{\mathcal{X}} = \mathbf{v}_{\mathcal{X}}$, that the classifier maps to c, in other words $|\{\mathbf{x} \in \mathbb{F} \mid \mathbf{x}_{\mathcal{X}}{=}\mathbf{v}_{\mathcal{X}} \text{ and } \kappa(\mathbf{x}){=}c\}|/|\{\mathbf{x} \in \mathbb{F} \mid \mathbf{x}_{\mathcal{X}}{=}\mathbf{v}_{\mathcal{X}}\}|$.

The *set of all WeakPAXp* for $\kappa(\mathbf{v}) = c$ wrt the threshold δ is denoted $\text{WeakPAXp}(\kappa, \mathbf{v}, c, \delta, \mathbb{F})$.

$\mathcal{X} \subseteq \mathcal{F}$ *is a PAXp* if it is a subset-minimal weak PAXp. The *set of all PAXp* for $\kappa(\mathbf{v}) = c$ wrt the threshold δ is denoted $\text{PAXp}(\kappa, \mathbf{v}, c, \delta, \mathbb{F})$.

The idea of using PAXp is to redefine the predicate to be studied for the next sub-sequence as we progress backwards. In order to fit the PAXp framework we define the classifier $\kappa_{s,\pi,p,d,k}$ as a binary classifier based on the utility of the state s. We note $u^k_{d,\pi,p}(s) = u_d(succ^k_{\pi,p}(\{(s, 1)\}))$, *the utility of* s wrt d given an horizon k, a policy π and a transition function p. The class of any state $\mathbf{x}$ is the result of a comparison between the utility of s and the utility of $\mathbf{x}$ for the respect of d.

Definition 4 (B-HXP classifier)**.** Given a state s, a policy π, a transition function p, a predicate d and a horizon k. The *B-HXP classifier*, denoted $\kappa_{s,\pi,p,d,k}$, is a function such that: for all $\mathbf{x} \in \mathcal{S}$,

$$\kappa_{s,\pi,p,d,k}(\mathbf{x}) = \begin{cases} True & \text{if } u^k_{d,\pi,p}(\mathbf{x}) \geq u^k_{d,\pi,p}(s) \\ False & \text{otherwise} \end{cases}$$

This classifier is specific to B-HXP. The utility threshold value depends on the state s which is the state associated with the most important action in the sub-sequence studied. It is used to generate a predicate d' which reflects a set of states at least as useful as s (with a probability of at least δ) for the respect of d. The predicate d' can then be seen as a sub-goal for the agent in order to satisfy d.

To assess whether a subset $\mathcal{X} \subseteq \mathcal{F}$ is a PAXp, it is necessary to calculate the utility of each state having this subset of features, which involves using the agent's policy π and the environment transition function p. A weak PAXp is then a sufficient subset of state features which ensures that a state utility is greater than or equal to the utility of s with probability at least δ. The new predicate is defined as the disjunction of every possible PAXp from s.

Definition 5. Given a state $s = (s_1, \ldots, s_n) \in \mathcal{S}$, a B-HXP classifier κ on $\mathcal{S}$, *the predicate PAXpred associated with* s for a given threshold δ is:

$$\text{PAXpred}_{\kappa}(s, \delta) = \bigvee_{\mathcal{X} \in \text{PAXp}(\kappa, s, True, \delta, \mathcal{S})} \left(\bigwedge_{f_i \in \mathcal{X}} f_i = s_i \right)$$

Example 1 (Cont)**.** *For this history,* δ *is set to* 1 *and* l *is* 4*. The first sub-sequence studied is: [nap, eat, water the plants, read]. The most important action relative to the achievement of "Bob is not hungry" is 'eat', with a score of, say,* 0.5 *and its associated state, say,* $(1, 0, 0, 1, 0)$*. We extract a PAXp, which includes the features* fridge *and* hungry *set to* 1*. This means that, whatever the values of the other features, a state with* fridge *and* hungry *set to* 1 *has (with* 100% *probability) a utility greater than or equal to* 0.5*. Now, we study the sub-sequence [work, shop,*

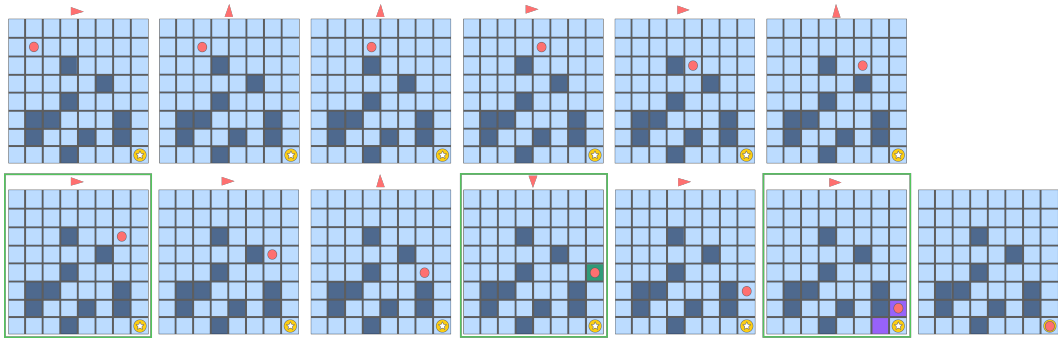

Figure 1: B-HXP for the *win* predicate. The agent is symbolized by a red dot, the dark blue cells are holes and the destination cell is marked by a star. Actions identified as important are highlighted by a green frame.

watch TV, nap], in which the most important action to achieve the new intermediate predicate is 'shop'.

In the backward analysis of H, the change of predicate allows us to look at a short-term objective to be reached, thus keeping the calculation of HXP reasonable. Our method is explained in pseudo-code in Algorithm 1. This algorithm allows us to go backwards through the history H, successively determining in each sub-sequence studied, the important action and its associated state predicate. The argmax function is used to find, in a given sub-sequence $H_{(i,j)}$, the most important action a, its associated state s, and its index in H. The latter is used to determine the next sub-sequence to consider. The *PAXpred* function is used to generate the new predicate to study in the next sub-sequence, based on Definition 5. The process stops when all actions have been studied at least once, or when the utility of the most important action in the current sub-sequence is 0. Finally, the algorithm returns a list of important actions and the different predicates found.

With B-HXPs, it is interesting to note that the number of actions to be presented to the user is not fixed. In the worst-case scenario, a user could end up with an explanation that refers to all actions as important. This would happen if it was always the last action in each subsequence which is the most important for achieving the current predicate. However, this problem was not observed in our experiments.

The computationally hard part of this approach is the predicate generation. Enumerating all the

Algorithm 1 B-HXP algorithm

Input: history H, maximal sub-sequence length l, agent's policy π, predicate d, transition function p, probability threshold δ, state space $\mathcal{S}$
Output: important actions A, predicates D

$A \leftarrow [\,]$; $D \leftarrow [\,]$; $u \leftarrow 1$
$i_{max} \leftarrow len(H)$; $i_{min} \leftarrow \max(0, i_{max} - l)$
while $i_{min} \neq 0$ and $u \neq 0$ **do**
  $i, s, a \leftarrow \underset{\substack{i \in [i_{min}, i_{max}] \\ (s,a) = H_i}}{\text{argmax}} \; imp^{l}_{d,\pi,p}(s,a)$
  $u \leftarrow u^{l}_{d,\pi,p}(s)$
  $d \leftarrow \text{PAXpred}_{\kappa_{s,\pi,p,d,l}}(s,\delta)$
  $A.append(a)$; $D.append(d)$
  $i_{max} \leftarrow i$; $i_{min} \leftarrow \max(0, i_{max} - l)$
end while
return A, D

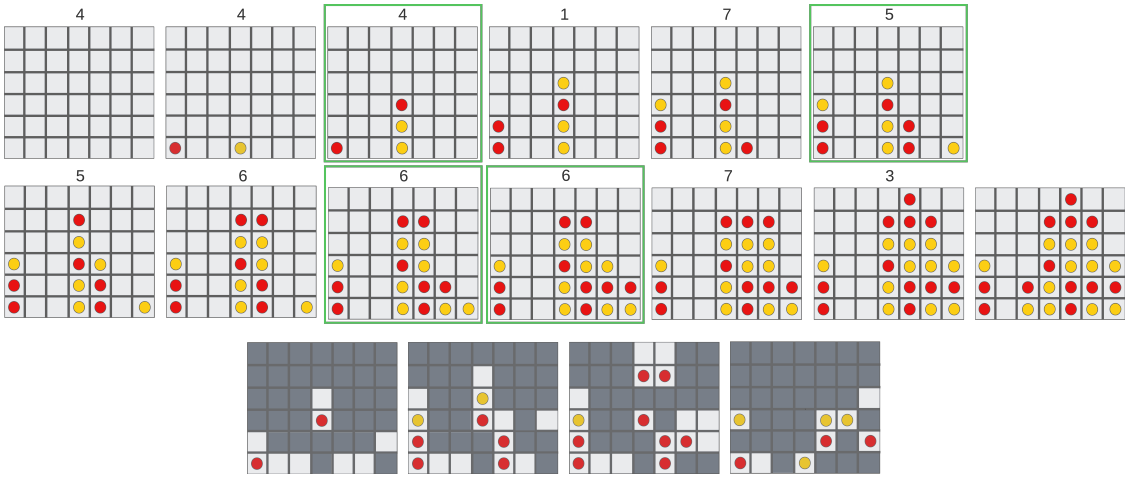

Figure 2: B-HXP for the *win* predicate. Above: the input history, showing 12 moves (where move = choice of column) of the agent (yellow) to which the environment (red) responds. Below: the predicates found by B-HXP corresponding to the four important moves it finds in the history, each highlighted by a green frame.

PAXp's turns out to be intractable. To support this assertion, even finding a single AXp (which is a PAXp with $\delta = 1$) is in general NP-hard, for example in the case of a DNF classifier [6]. Also, finding a single PAXp when $\delta < 1$ is NP-hard even for decision trees [3]. A further computational difficulty, specific to our problem, is that our classifier $\kappa_{s,\pi,p,d,k}$ requires, at each call, the computation of the action utility, which is a #W[1]-hard problem [21] w.r.t. the parameter k.

Thus, we decided to limit the definition of a predicate d to the generation of one weak PAXp. To obtain a predicate d in reasonable time, we need to look at a particular class of weak PAXp, the *locally-minimal* PAXps, which are not necessarily subset-minimal. Formally, a set of features $\mathcal{X} \subseteq \mathcal{F}$ is a *locally-minimal* PAXp if $\mathcal{X} \in \text{WeakPAXp}(\kappa, \mathbf{v}, c, \delta, \mathbb{F})$ and for all $j \in \mathcal{X}$, $\mathcal{X} \setminus \{j\} \notin \text{WeakPAXp}(\kappa, \mathbf{v}, c, \delta, \mathbb{F})$ The *findLmPAXp* algorithm [13] is used to calculate a *locally-minimal* PAXp.

In short, B-HXP keeps the calculation of importance scores exhaustive, by cutting the length-k history into sub-sequences of length l, starting with the end. The most important action of a sub-sequence is retained, and its associated state is used to define d', the new predicate to study, using *locally-minimal* PAXp. d' is then studied in a new sub-sequence. This process is iterated throughout the history. The next section presents examples of B-HXP.

4 Experiments

The experiments were carried out on 3 RL problems: Frozen Lake (FL), Connect4 (C4) and Drone Coverage (DC) [20]. Q-learning was used to solve the FL problem, and Deep-Q-Network for C4 and DC. The agents' training was performed using a Nvidia GeForce GTX 1080 TI GPU, with 11 GB of RAM. The B-HXP examples were run on an HP Elitebook 855 G8 with 16GB of RAM (source code available at: `https://github.com/lsaulier/HXP`).

The first part of this section describes the problems and the studied predicates (for more details, see [21]). The second part presents B-HXP examples.

In the figures, the history is displayed over two lines. The action taken by the agent from a state is shown above it. Important actions and their states are highlighted by a green frame. The third line of figures 2 and 3 corresponds to the predicates generated during the B-HXP, where a dark grey cell means that this feature is not part of the predicate. In Tables 1,

2 and 3, the importance scores given are w.r.t. either the initial predicate or the intermediate ones.

4.1 Description of the problems

In FL, the agent moves on the surface of a frozen lake (2D grid) to reach a certain goal position, avoiding falling into the holes. The agent can move in any of the 4 cardinal directions. However, due to the slippery surface of the frozen lake, the agent may slip and not end up in the position induced by the chosen action. Indeed, if the agent chooses a direction (e.g. *up* as in the second state of Figure 1), it has 0.6 probability to go in this direction and 0.2 to go towards each remaining direction except the opposite one (e.g., for *up*, 0.2 to go *left* and 0.2 to go *right*, as occurred in the scenario of Figure 1 where the agent moved right in the third state after performing *up* in the second one). The agent's state is composed of 5 features: its position (P) and previous position (PP) on the map, the position of one of the two holes closest to the agent (HP), the Manhattan distance between the agent's initial position and his current position (PD), and the total number of holes on the map (HN). Predicates *win*, *holes* and *region* were studied. They respectively determine whether the agent reaches the goal, falls into a hole or reaches a pre-defined set of map positions.

The C4 game is played on a 6 by 7 vertical board, where the goal is to align 4 tokens in a row, column or diagonal. Two players play in turn. An agent's state is the whole board. 5 predicates were studied: *win*, *lose*, *3 in a row*, *prevent 3 in a row* and *control mid-column*.

In DC, four drones must cover (observe) the largest area of a windy 2D map, while avoiding crashing into a tree or another drone. A drone can move in any of the 4 cardinal directions or not. A drone cover is a 3×3 square centered on it. A cover for a drone is optimal when it does not contain any trees and there is no overlap with the cover of the other drones. An agent's state is made up of its view, a 5×5 image centered on it, and its position, represented by (x, y) coordinates. Ten predicates for the DC problem were studied (local and global versions of): *perfect cover, maximum reward, no drones, crash* and *region*. Local versions concern a single agent, whereas the global versions concern all agents.

Table 1: Importance scores in the FL history 1

Predicate	Time-step / Importance score			
win	8 -0.001	9 0.04	10 0.012	11 **0.114**
$\text{PAXpred}_{\kappa}(s_{11}, 0.7)$ (a.k.a. *purple)*	7 0.006	8 -0.008	9 **0.102**	10 0.087
$\text{PAXpred}_{\kappa}(s_{9}, 0.7)$ (a.k.a. *green)*	5 -0.0003	6 **0.0**	7 -0.001	8 -0.0003

4.2 B-HXP examples

To provide B-HXPs in reasonable time, the $sample$ parameter, which corresponds to the maximum number of states observed for a feature evaluation in the *findLmPAXp* algorithm [13], i.e. the predicate generation, was set to 10 in the following examples. In other words, to avoid an exhaustive search over $\mathbb{F}$, the proportion in Definition 3 was computed based on 10 samples.

A B-HXP (computed in 2 seconds) for a FL history is shown in Figure 1, with l=4, δ=0.7. Importance scores are presented in Table 1. The *right* action linked to the state $s_{11} = \{P = (7,8), PP = (6,8), HP = (6,7), PD = 13, HN = 10\}$ is the most important in the first sub-sequence studied in order to *win*. The predicate, named *purple*, computed from s_{11} with $\delta = 0.7$ is $PAXpred_{\kappa}(s_{11}, 0.7) = \{PD = 13\}$. The states described by the predicate are shown in purple in Figure 1. In the following sub-sequence, the *down* action linked to state $s_9 = \{P = (5,8), PP = (5,7), HP = (6,7), PD = 11, HN = 10\}$ is the most important to respect *purple*. The predicate, named *green*, computed based on s_9 is $PAXpred_{\kappa}(s_9, 0.7) = \{P = (5,8), PP = (5,7), HP = (6,7)\}$ (the states described are shown in green in Figure 1). A predicate is generic if it is respected by a large number of different states. We note that *purple* describes more states than *green*. The latter is not generic enough, which is reflected in the importance scores, which are close to 0: whatever the action, it is unlikely to respect this predicate after 4 time steps. The entire history is not explored when calculating the B-HXP, as the utility

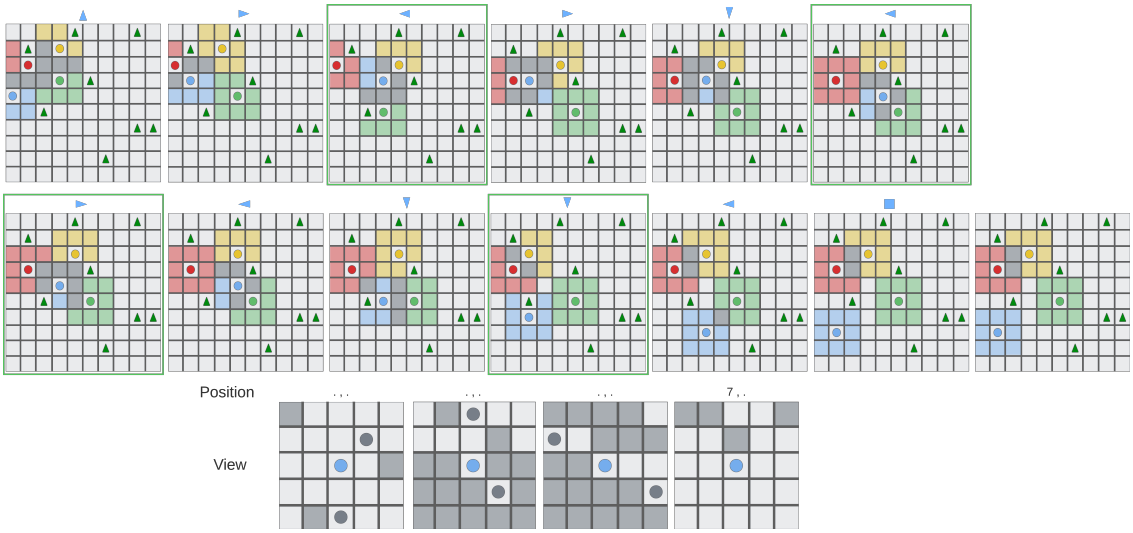

Figure 3: B-HXP for the *perfect cover* predicate. The intermediate predicates are shown in the last row. The rightmost of these states that the blue drone is in row 7 and that the light grey squares are free. The other intermediate predicates impose the relative positions of two other drones and that some squares are free. The drones are represented by dots and the trees by green triangles. In the history, a colored cell means that the area is covered by the drone of the same color and a dark grey cell indicates an overlap of the coverage of different drones.

Table 2: Importance scores in the C4 history 2

Predicate	Time-step / Importance score		
win	9 **0.726**	10 0.099	11 0.16
$\text{PAXpred}_{\kappa}(s_9, 0.8)$	6 -0.006	7 0.03	8 **0.113**
$\text{PAXpred}_{\kappa}(s_8, 0.8)$	5 **0.003**	6 0.0	7 0.0
$\text{PAXpred}_{\kappa}(s_5, 0.8)$	2 **0.003**	3 0.0	4 0.0

of the last state selected s_6 is 0. The selected actions form a meaningful explanation when we look at the predicates studied. However, the redefined predicates fairly quickly become very specific and probably of little help in explaining why the agent won.

With l set to 3 and δ to 0.8, a B-HXP (computed in 10 seconds) for a C4 history is shown in Figure 2. A large part of the board is ignored in the predicates, which gives the user an intuition of the type of states that the agent must reach. Importance scores are presented in Table 2. Almost all the actions returned are related to setting up the token alignment leading to victory, which is interesting because the predicate *win* is only studied on the last three states of the history. The other predicates provide actions linked to achieving a *win*. The first redefined predicate (the rightmost image on the third line of Figure 2) describes a partial board configuration: from positions satisfying this predicate, an agent following the learnt policy has at least 80% chance of achieving a win in the final position. However, as with FL, apart from the predicate defined at time-step 9, the other predicates generated seem to be not sufficiently generic, which can be seen in the scores. Indeed, the second redefined predicate (the second-from-right image in the last line of Figure 2) is so specific as to uniquely determine the exact board configuration (given the height of columns and the number of tokens played).

Table 3: Importance scores in the DC history 3

Predicate	Time-step / Importance score		
perfect cover	9 **0.114**	10 0.063	11 0.056
$\text{PAXpred}_{\kappa}(s_9, 1)$	6 **0.008**	7 0.006	8 0.002
$\text{PAXpred}_{\kappa}(s_6, 1)$	3 0.053	4 -0.013	5 **0.09**
$\text{PAXpred}_{\kappa}(s_5, 1)$	2 **0.034**	3 -0.036	4 0.023

The value of l has an important impact on the computation of importance scores. This point is explored in more detail in the conclusion.

A B-HXP (calculated in 13 seconds) for a DC history is shown in Figure 3, with l=3 and δ=1. The explanation is for the blue drone with the original predicate that this agent has a perfect cover. The importance scores are presented in Table 3. The predicates give a good intuition of the type of state (position and 5×5 view) the agent is trying to reach.

5 Related Work

XRL methods can be clustered according to the scope of the explanation (e.g. explaining a decision in a given situation or the policy in general), the key RL elements used to produce the explanation (e.g. states [10, 18], rewards [14, 1]), or the form of the explanation (e.g. saliency maps [10] or sequence-based visual summaries [2, 22, 20]).

One approach consists in generating counterfactual trajectories (state-action sequences) and comparing them with the agent's trajectory. In [1], reward influence predictors are learnt to compare trajectories. The counterfactual one is generated based on the user's suggestion. In [25], a contrastive policy based on the user's question is produced to generate the counterfactual trajectory. In the MDP context, Tsirtsis et al. generate optimal counterfactual trajectories that differ at most by k actions [24].

EDGE [11] is a self-explainable model. Like HXP, it identifies the important elements of a sequence. However, EDGE is limited to importance based on the final reward achieved, whereas HXPs allow the study of various predicates. In addition, HXP relies on the transition function (which is assumed to be known) and the agent's policy to explain, whereas EDGE [11] requires the learning of a predictive model of the final-episode reward.

6 Conclusion

Our paper is a follow-up to the work carried out in [21]. HXP is a method that makes it possible to answer, for a given history, the question: *"Which actions were important to ensure that the predicate d was achieved, given the agent's policy π?"*. To do this, an importance score is computed for each action in the history. To provide explanations for long histories, without importance score approximation, we defined an approach named Backward-HXP. Starting from the end of the history, this involves iteratively studying a subsequence, highlighting the most important action in it and defining a new intermediate predicate to study for the next sub-sequence. The intermediate predicate is a *locally-minimal PAXp* associated to the state where the most important action took place.

In the experiments, we observed that the genericity of a predicate d and the search horizon l influence the importance scores. The more generic the predicate d, the greater the probability of finding states with a l horizon that respect d. Thus, the importance scores generated are significant, regardless of l. Conversely, a less generic predicate makes it more difficult to evaluate an action. In this case, the value of l is discriminating. A too specific predicate d can lead to insignificant importance scores for the respect of d, as the utility of actions is close to 0. In several histories, notably C4 ones, the predicates generated are non generic, leading to less interesting explanations.

Although in the examples the important actions are often related to the respect of the initial predicate, this is not always the case. If we consider the first redefined predicate as a possible cause of the predicate being satisfied in the final state, then the second redefined predicate is a possible cause of a possible cause. The notion of causality can quickly

become highly diluted (due to the fact for computational reasons, we study a single cause at each step). The user must be aware of this effect when computing B-HXPs.

HXP and B-HXP offer the user great diversity in the study of agent behavior through its notion of predicate. Furthermore, this approach is agnostic with regard to the agent learning algorithm. As described in [21], the strong assumption for the use of HXP and B-HXP is the knowledge of the transition function. It must be known during the explanation phase (not necessarily during training), or at least approximated, for example using an RL model-based method.

More experiments are needed to ensure the quality and scalability of B-HXP, specially in environments with a large number of transitions. When calculating a *locally-minimal* PAXp, the order in which features are processed is important. A future work would be to direct the generation of *locally-minimal* PAXp using a feature ordering heuristic, such as LIME [19]. In this way, it would be possible to compare the intermediate predicates and check whether this changes the important actions returned.

Our experiments have shown the feasibility of the finding important actions in a long sequence of actions by redefining predicates, working backwards from the end of the sequence. However, we found that the intermediate predicates can quickly become very specific leading to the difficulty of calculating the importance scores of actions w.r.t. these very specific predicates. Further research is required to investigate this point.

Acknowledgement

We would like to thank the reviewers for their pertinent comments, which helped to improve the paper quality.

References

[1] Amal Alabdulkarim and Mark O. Riedl. Experiential explanations for reinforcement learning. *CoRR*, abs/2210.04723, 2022.

[2] Dan Amir and Ofra Amir. HIGHLIGHTS: summarizing agent behavior to people. In *AAMAS*, pages 1168–1176. ACM, 2018.

[3] Marcelo Arenas, Pablo Barceló, Miguel A. Romero Orth, and Bernardo Subercaseaux. On computing probabilistic explanations for decision trees. In *NeurIPS*, 2022.

[4] Lijia Chen, Pingping Chen, and Zhijian Lin. Artificial intelligence in education: A review. *IEEE Access*, 8:75264–75278, 2020.

[5] Jeffery A. Clouse. *On integrating apprentice learning and reinforcement learning*. PhD thesis, UMass Amherst, 1996.

[6] Martin C. Cooper and João Marques-Silva. Tractability of explaining classifier decisions. *Artif. Intell.*, 316:103841, 2023.

[7] Adnan Darwiche. Human-level intelligence or animal-like abilities? *Commun. ACM*, 61(10):56–67, 2018.

[8] Ngozi Clara Eli-Chukwu. Applications of artificial intelligence in agriculture: A review. *Engineering, Technology & Applied Science Research*, 9(4), 2019.

[9] European Commission. Artificial Intelligence Act, 2021.

[10] Samuel Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. Visualizing and understanding Atari agents. In *ICML*, pages 1787–1796. PMLR, 2018.

[11] Wenbo Guo, Xian Wu, Usmann Khan, and Xinyu Xing. EDGE: explaining deep reinforcement learning policies. In *NeurIPS*, pages 12222–12236, 2021.

[12] Pavel Hamet and Johanne Tremblay. Artificial intelligence in medicine. *Metabolism*, 69:S36–S40, 2017.

[13] Yacine Izza, Xuanxiang Huang, Alexey Ignatiev, Nina Narodytska, Martin C. Cooper,

and João Marques-Silva. On computing probabilistic abductive explanations. *Int. J. Approx. Reason.*, 159:108939, 2023.

[14] Zoe Juozapaitis, Anurag Koul, Alan Fern, Martin Erwig, and Finale Doshi-Velez. Explainable reinforcement learning via reward decomposition. In *IJCAI/ECAI workshop on explainable artificial intelligence*, page 7, 2019.

[15] Zachary C. Lipton. The mythos of model interpretability. *Commun. ACM*, 61(10):36–43, 2018.

[16] Stephanie Milani, Nicholay Topin, Manuela Veloso, and Fei Fang. A survey of explainable reinforcement learning. *CoRR*, abs/2202.08434, 2022.

[17] White House Office of Science and Technology. Blueprint for an AI Bill of Rights. 2022.

[18] Matthew L. Olson, Lawrence Neal, Fuxin Li, and Weng-Keen Wong. Counterfactual states for Atari agents via generative deep learning. *CoRR*, abs/1909.12969, 2019.

[19] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. In *SIGKDD*, pages 1135–1144. ACM, 2016.

[20] Léo Saulières, Martin C. Cooper, and Florence Bannay. Reinforcement learning explained via reinforcement learning: Towards explainable policies through predictive explanation. In *ICAART, Vol. 2*, pages 35–44, 2023.

[21] Léo Saulières, Martin C Cooper, and Florence Dupin de Saint Cyr. Predicate-based explanation of a Reinforcement Learning agent via action importance evaluation. In *ECML/PKDD workshop AIMLAI*, 2023.

[22] Pedro Sequeira and Melinda T. Gervasio. Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations. *Artif. Intell.*, 288:103367, 2020.

[23] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[24] Stratis Tsirtsis, Abir De, and Manuel Rodriguez. Counterfactual explanations in sequential decision making under uncertainty. In *NeurIPS*, pages 30127–30139, 2021.

[25] Jasper van der Waa, Jurriaan van Diggelen, Karel van den Bosch, and Mark A. Neerincx. Contrastive explanations for reinforcement learning in terms of expected consequences. *CoRR*, abs/1807.08706, 2018.