Backward explanations via redefinition of predicates

Léo Saulières
IRIT
University of Toulouse III
Toulouse, France
leo.saulieres@irit.fr

Martin C. Cooper
IRIT
University of Toulouse III
Toulouse, France
martin.cooper@irit.fr

Florence Dupin de Saint-Cyr
IRIT
University of Toulouse III
Toulouse, France
florence.bannay@irit.fr

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 [20]. 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 show the ability of B-HXP to summarise long histories.

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, 3]. 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 [24, 9] and researchers [6, 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 "state importance" [4]. 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

17th European Workshop on Reinforcement Learning (EWRL 2024).

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 [20], consists of studying a history of agent interactions with the environment through the prism of a certain predicate, a history being a sequence of pairs (state, action). A predicate d is any boolean function of states which is true in the final state of the history.This XRL method answers the question: *"Which actions were important to ensure that d was achieved, given the agent's policy π?"*. This paper follows this paradigm of explanation of a history with respect to a predicate, by proposing a new way of defining the important actions of a history, called Backward-HXP (B-HXP). In particular, B-HXP was investigated because of the limits of (forward) HXP in explaining long histories due to the $\#$W[1]-hardness of HXP [20].

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 result carried out on the classic *Frozen Lake* environment. Section 5 presents related works and Section 6 concludes.

2 History eXplanation via Predicates (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 that can be performed from s. $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ and $p : \mathcal{S} \times \mathcal{A} \rightarrow Pr(\mathcal{S})$ 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. $\pi : \mathcal{S} \rightarrow \mathcal{A}$ 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. Due to the Markovian nature of the process, transitions at different instants are independent and hence the probabilities given by the transition function p can be multiplied when calculating the probability of a scenario (sequence of states). In the following, we use the function $next$, based on the policy π and the transition function described by p, to compute the next possible states (associated with their probabilities) given a set S of (state, probability) pairs: $next_{\pi,p}(S) = \{(s', pr \times p(s'|s, a)) : (s, pr) \in S, a = \pi(s) \text{ and } p(s'|s, a) \neq 0\}$. In order to compute the set of final states reachable at horizon k from a set of states S, using the agent's policy π and the transition function p, the function $succ^k_{\pi,p}$ is defined recursively by $succ^0_{\pi,p}(S) = S$ and $succ^{n+1}_{\pi,p}(S) = next_{\pi,p}(succ^n_{\pi,p}(S))$. As a remark for the implementation of *next*, it is worth noting that the result of *next* may contain several occurrences of the same state (with possibly different probabilities), if several paths lead to it from the set of current states. An optimisation would consist in merging the equal states of a set, by summing their associated probabilities.

HXPs [20] 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. The language used for the predicate is based on the features that characterize a state. Each feature f_i has a range of values defined by a domain D_i, the set of all features is denoted $\mathcal{F} = \{f_1, ..., f_n\}$. 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. A state $s \in S$ is an interpretation in the language based on the vocabulary $(l_{i,j})_{i \in [1,n], j \in D_i}$ such that $s(l_{i,j}) = True$ if and only if the feature i as the value j in the state s. It follows that the predicate d can be evaluated in linear time for a given state. 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 by:

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:*

$$imp^k_{d,\pi,p}(s,a) = u_d(succ^k_{\pi,p}(S_{(s,a)})) - \underset{a' \in A(s)\setminus\{a\}}{\text{avg}} u_d(succ^k_{\pi,p}(S_{(s,a')}))$$

where avg *is the average and* $S_{(s,a)}$ *is the support of* $p(.|s,a)$*.*[1]

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 [20], so it is necessary to approximate it, in particular by generating only part of the length-k scenarios with the *succ* function.

To handle long histories on problems where the number of possible transitions is large, i.e. large horizon k and large branching factor (denoted b hereafter), 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 explaining histories in a step-by-step backward approach, which allows us to provide explanations in reasonable time for long histories, without having to approximate the calculation of scores. 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 history explanation.

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 s to define the new predicate to be studied. The process then iterates treating s as the final state with this new predicate. 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. Actions are then evaluated with respect to the new predicate 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 final 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 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 in the history 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 the B-HXP method in detail, we introduce some notations. $H = (s_0, a_0, s_1, ..., a_{k-1}, s_k)$ 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 $H_{(i,j)} = (s_i, a_i, ..., s_j)$. 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

[1] i.e. $S_{(s,a)} = \{(s', p(s'|s,a)) \mid p(s'|s,a) \neq 0\}$

these features guarantees (with a 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: $\kappa : \mathbb{F} \to \mathcal{K}$. 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 $|\{\mathbf{x} \in \mathbb{F} \mid \mathbf{x}_{\mathcal{X}}=\mathbf{v}_{\mathcal{X}}$ and $\kappa(\mathbf{x})=c\}|/|\{\mathbf{x} \in \mathbb{F} \mid \mathbf{x}_{\mathcal{X}}=\mathbf{v}_{\mathcal{X}}\}|$.

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

The idea is to use PAXp's to redefine the predicate to be studied for the next sub-sequence as we progress backwards. In order to fit into 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 $\kappa_{s,\pi,p,d,k}(\mathbf{x})$ 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:*

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

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. Indeed, this is the only PAXp, so PAXPred is (*fridge=*1* $\wedge$ hungry=*1). Now, we study the sub-sequence [work, shop, 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. It is a polynomial-time algorithm (considering the length of the sub-sequence l as a constant) which performs a polynomial number of importance score and predicate computations (depending in particular on l and the importance scores obtained along the execution). However, calculating the importance score and the predicate are costly operations, as we will show in the following.

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 ← [] ; D ← []; u ← 1
i_max ← len(H) ; i_min ← max(0, i_max − l)
while i_min ≠ 0 and u ≠ 0 do
    i, s, a ← argmax_{i ∈ [i_min, i_max], (s, a) = H_i}  imp^l_{d,π,p}(s, a)
    u ← u^l_{d,π,p}(s)
    d ← PAXpred_{κ_{s,π,p,d,l}}(s, δ)
    A.append(a) ; D.append(d)
    i_max ← i ; i_min ← max(0, i_max − l)
end while
return A, D
```

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.

B-HXP provides the user with an explanation even for long histories. Our motivation behind B-HXP is to reduce the complexity of calculating important actions in comparison with forward HXP. In the remainder of this section, we justify the choices made in defining B-HXP by analysing in detail the theoretical complexity of certain subproblems encountered by HXP and B-HXP.

Proposition 1 *Given a policy π, a transition function p and predicate d, the importance score computation of an action a at horizon k from any state s for the respect of d, $imp^k_{d,\pi,p}(s,a)$, is #P-hard.*

Proof. This complexity is a direct consequence of the result in [20]: given the length of the search horizon k as a parameter, calculating the importance of an action is #W[1]-hard. □

To avoid this computational complexity, the search horizon l is chosen to be a small constant in the B-HXP calculation of importance scores. Using b to denote the maximum number of successor states from any given state (i.e. $b = \max_{s\in\mathcal{S}} |\{s' \in \mathcal{S} : p(s|s,\pi(s)) > 0\}|$), there are a maximum of b^l scenarios generated, and hence we have the following lemma.

Lemma 1 *Given a policy π and a transition function p, we assume that b, the maximum number of successor states is a polynomial function of the instance size (i.e. the number of bits necessary to specify the history together with π, p and the predicate d). For a constant search horizon l, the computation of the importance score of any action a from any state s for the respect of d ($imp^l_{d,\pi,p}(s,a)$) and the utility of the state s are in time which is polynomial in the size of the instance.*

However, in comparison with forward HXP, it is necessary to compute intermediate predicates, in particular with PAXpred (Definition 5). It is interesting to note that the B-HXP classifier (Definition 4) κ used in PAXpred can be evaluated in polynomial time because it is based on the calculation of the utility of a state (the main component in the importance score computation). Unfortunately, from a state s, the number of PAXp's according to κ, can be exponential in the size of s.

Lemma 2 *Given a state s, in the worst case, PAXpred returns a predicate of size which may be exponential in the size of s.*

To support this assertion, consider the following example.

Example 2 *A state consists of n features $f_1, \ldots, f_n$, initially all set to 0. An agent's ith action consists in assigning a value to f_i from its domain D_i, where $D_i = \{0,1\}$ ($i = 1, \ldots, n-1$) and $D_n = \{0,1,n\}$. The transition function is deterministic and the agent obtains a reward only by reaching any state s^{goal} such that $\sum_{i=1}^{n} f_i \geq n + \frac{n-1}{2}$. Let the predicate d be* goal*, i.e. a predicate checking whether the agent reaches a state s^{goal}. Consider the state $s = (1, \ldots, 1, 0)$ after the agent has made $n-1$ assignments. Clearly, assigning n to f_n establishes the predicate in one step. Moreover, any state with at least (n-1)/2 assignments of 1 among the first $n-1$ features, allows us to attain the goal in one step by assigning n to the last feature f_n. Hence, the PAXp's of this predicate d from s and for $l = 1$, $\delta = 1$ are precisely the subsets of $(n-1)/2$ literals of the form $f_i = 1$ ($1 \leq i \leq n-1$). There are $\binom{n-1}{(n-1)/2}$ such PAXp's, so the corresponding predicate PAXpred (for $\delta = 1$) is of exponential size.*

To exhaustively determine an intermediate predicate *PAXpred*, therefore, exponential space in the size of the state s is required. One approximation to *PAXpred* is to calculate only one AXp (i.e. a PAXp with $\delta = 1$). This reduces the problem to a more amenable co-NP problem [5] where only a counterexample state s' is needed to show that the set of features $\mathcal{X} \subseteq \mathcal{F}$ is not an AXp. Unfortunately, this approximation yields predicates that are often too specific because of $\delta = 1$. Instead, in order to provide sparser intermediate predicates, the considered approximation consists in computing one PAXp, or more precisely one *locally-minimal* PAXp. *Locally-minimal* PAXp's is a particular class of weak PAXp 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. Although the approximation consists of calculating just one LmPAXp, it remains a hard counting problem. We first prove hardness before explaining why this is nevertheless an improvement over the hardness of forward HXP.

Lemma 3 *Given a state s, the computation of a locally-minimal PAXp is in FP$^{\#P}$ (i.e. in polynomial time using a #P-oracle) and determining whether a given subset of features is a locally-minimal PAXp is #P-hard.*

Proof. We first show the inclusion in FP$^{\#P}$. From a specific state s, the search for a locally-minimal PAXp $\mathcal{X} \subseteq \mathcal{F}$ begins by initialising $\mathcal{X}$ to $\mathbf{F}$ (the trivially-correct explanation consisting of all the features). Then, the following test is carried out for a feature $f_i \in \mathcal{X}$: if f_i is removed from $\mathcal{X}$, does $\mathcal{X}$ remain a WeakPAXp? If so, f_i is removed, otherwise retained. This test is performed for each feature of $\mathcal{X}$. Determining whether a subset $\mathcal{X} \subseteq \mathcal{F}$ is a WeakPAXp amounts to counting the number of states $\mathbf{x}$ which match s on $\mathcal{X}$ (i.e. $\mathbf{x}_{\mathcal{X}} = s_{\mathcal{X}}$) that are classified as True (i.e. such that $\kappa_{x,\pi,p,d,k}$ =True). Lemma 1 tells us that κ can be evaluated in polynomial time, so WeakPAXp belongs to #P. The search for a locally-minimal PAXp can thus be performed in polynomial time in the size of s, i.e. its number of features, using a #P-oracle WeakPAXp.

To prove #P-hardness, it suffices to give a polynomial reduction from PERFECT MATCHINGS which is a #P-complete problem [26]. Consider a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges. V is decomposed into two parts V_1 and V_2 so that each edge has one end in V_1 and another in V_2. In other words, G is a bipartite graph. A perfect matching is a set of edges such that no pairs of vertices has a common vertex.

Let us define a set of n features $f_i \in \mathcal{F}$ such that each feature f_i corresponds to a vertex, say i, in V_1. Let us define the domain D_i of each feature f_i as the set of vertices in V_2 that are related to the vertex i by E. In other words, an edge $e \in E$ is a possible assignment of a feature f_i to a value v_j. The feature space $\mathbb{F}$ is modeled by G. Indeed, each point $\mathbf{x} = (f_1, \ldots, f_n) \in \mathbb{F}$ can be obtained by assigning each feature in V_1 to a value in V_2. Thus, a state $\mathbf{x}$ is represented by a set of edges $E_{\mathbf{x}} \subseteq E$. To compute WeakPAXp, the following classifier is used:

$$\kappa(\mathbf{x}) = \begin{cases} True & \text{if } \forall i \neq j, f_i \neq f_j \\ False & \text{otherwise} \end{cases}$$

This classifier outputs True for any state $\mathbf{x}$ comprising a distinct valuation for each feature. In G, such a state $\mathbf{x}$ is represented by a set of edges $E_{\mathbf{x}} \subseteq E$ in such a way that each pair of edges in $E_{\mathbf{x}}$ has no common vertices. Thus, $E_{\mathbf{x}}$ is a perfect matching. From a specific point $\mathbf{v}$ such that $\kappa(\mathbf{v}) = True$, we are interested in knowing whether $\mathcal{X} = \emptyset$ is a weakPAXp, for different values of δ. Based on Definition 3, it is necessary to calculate the proportion of states classified as True to determine whether $\mathcal{X}$ is a weakPAXp. With $\mathcal{X} = \emptyset$, the total number of states that match $\mathcal{X}$ corresponds to the number of states in $\mathbb{F}$. This is easily obtained by multiplying the domain-sizes $|D_i|$, $|\mathbb{F}| = \Pi_{i=1}^{n} |D_i|$. The number of states which match $\mathcal{X}$ and are classified by κ to True, is obtained by counting the states which have a distinct valuation for each feature, i.e. by counting the perfect matchings in G. $\mathcal{X} = \emptyset$ is a weakPAXp iff it is a (locally-minimal) PAXp. Thus, we have reduced the PERFECT MATCHINGS problem to a polynomial number of calls to the locally-minimal PAXp problem. Hence there is a polynomial-time Turing reduction from PERFECT MATCHINGS to the locally-minimal PAXp problem, which is therefore #P-hard. □

Consequently, the B-HXP complexity is deduced from the complexity of the computation of importance scores and the generation of the intermediate predicates.

Proposition 2 *The B-HXP computation is in FP$^{\#P}$ and is #P-hard.*

Proof. From Lemma 1 and Lemma 3, we deduce that the computational complexity of B-HXP is in FP$^{\#P}$ and is #P-hard, in particular due to the complexity of the generation of a new predicate. □

The computation of a B-HXP and the importance score computation in forward HXP are both #P-hard. However, it is important to note that counting a number of states (in B-HXP) is less computationally expensive than counting a number of scenarios (i.e. a sequence of k states).

We can provide a finer analysis by studying fixed-parameter tractability in n, the size of a state. A problem is fixed parameter tractable (FPT) with respect to parameter n if it can be solved by an algorithm running in time $O(f(n) \times N^h))$, where f is a function of n independent of the size N of the instance, and h is a constant [7].

Proposition 3 *Given a sequence of k states, with each state of size n, a policy π, a predicate d, a transition function p, a threshold δ, a constant length l and b, the maximum number of successor states from any given state using π which is assumed to be polynomial in the instance's size, the complexity of finding a B-HXP is FPT in n.*

Proof. Recall that the input includes a length-k history made up of k states of size n. An exhaustive search over all possible alternative length-k scenarios would require $\Omega(nb^k)$ time (and hence is not FPT in n because of the exponential dependence on k, even if b is a constant). On the other hand, B-HXP need only exhaust over scenarios of constant length l. The complexity of redefining the predicate by finding one LmPAXp is $f(n)b^l$ for some function f of n (the state size). This redefinition of the predicate must be performed at most k times, giving a complexity in $O(f(n)b^l k)$. It follows that B-HXP is FPT in n since $b^l k$ is a polynomial function of the size of an instance. □

In short, B-HXP keeps the calculation of importance scores exhaustive, by cutting the length-k history into sub-sequences of length l, starting from 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.

The calculation of new predicates is computationally challenging but is feasible (using certain approximations: calculating just one rather than all PAXp's and, as we will see later, the use of sampling rather than exhaustive search over feature space). The resulting method B-HXP provides a novel approach to explaining histories.

4 Experiments

The experiments were carried out on 3 RL problems: Frozen Lake (FL), Connect4 (C4) and Drone Coverage (DC) [21]. Q-learning [28] was used to solve the FL problem, and Deep-Q-Network [17] 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 (a link to the source code will be made available in the final version).

The first part of this section describes FL and the studied predicates. The second part presents a B-HXP for the predicate win in the FL history example presented in Figure 1. Description of C4 and DC together with additional B-HXP examples can be found in Appendix A. In the figure, 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. In Table 1 the importance scores given are w.r.t. either the initial predicate or the intermediate ones.

In FL, the agent (symbolized in Figure 1 by a red dot) moves on the surface of a frozen lake (2D grid) to reach a certain goal position (the one marked with a star), avoiding falling into the holes (symbolized by dark blue cells). The agent can move in any of the 4 cardinal directions. However, due to the slippery surface of the frozen lake, if the agent chooses a direction (e.g. *down*), 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 *down*, 0.2 to go *left* and 0.2 to go *right*). 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). This last feature was added as a check: since it is a constant it should never appear in the redefined predicate, which was indeed the case. 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.

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.

Table 1: Importance scores in the FL history 1

Predicate	Time-step / Importance score			
win	8	9	10	11
	-0.001	0.04	0.012	**0.114**
$\text{PAXpred}_\kappa(s_{11}, 0.7)$	7	8	9	10
(a.k.a. *purple*)	0.006	-0.008	**0.102**	0.087
$\text{PAXpred}_\kappa(s_9, 0.7)$	5	6	7	8
(a.k.a. *green*)	-0.0003	**0.0**	-0.001	-0.0003

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 penultimate 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 $PAXpred_\kappa(s_{11}, 0.7) = (PD = 13)$, computed from s_{11} with $\delta = 0.7$, is named *purple* hereafter. 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 $PAXpred_\kappa(s_9, 0.7) = (P = (5,8)) \wedge (PP = (5,7)) \wedge (HP = (6,7))$, computed from s_9, is named *green* (the states described are shown in green in Figure 1). A predicate is said to be *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 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.

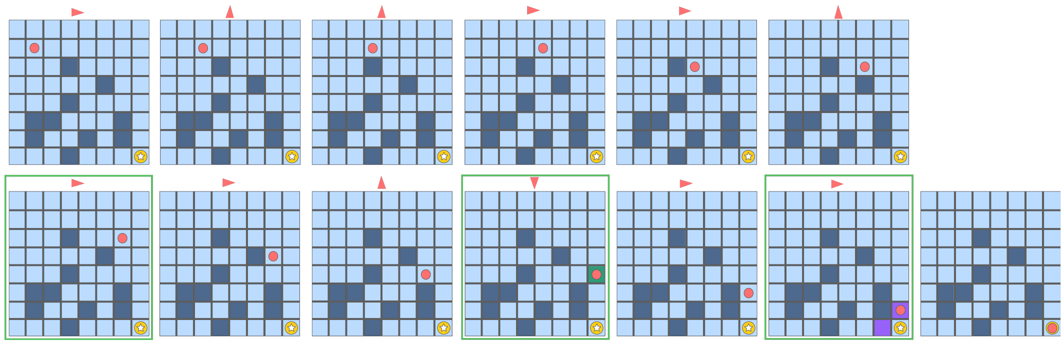

Figure 1: An example of history with 13 states $s_0, \ldots, s_{12}$ for the FL problem on which a B-HXP is computed for the *win* predicate (where important actions are highlighted by a green frame)

.

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] sequence-based visual summaries [2, 22, 21]).

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 [27], a contrastive policy based on the user's question is built to produce the counterfactual trajectory. In the MDP context, Tsirtsis et al. generate optimal counterfactual trajectories that differ at most by k actions [25]. The importance score used in this paper is in line with the counterfactual view by evaluating scenarios where a different action took place at a given time.

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

HXP (History eXplanation via Predicates) [20] is a paradigm that, for a given history, answers 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. Unfortunately this calculation is #W[1]-hard with respect to the length of the history to explain. To provide explanations for long histories, without resorting to importance score approximation, we propose the Backward-HXP approach: starting from the end of the history, it iteratively studies a subsequence, highlighting the most important action in it and defining a new intermediate predicate to study for the next sub-sequence (working backwards). In this paper, we call "locally-minimal PAXp" a particular Probabilistic Abductive eXplanation which is a minimal set of features for which the probability of achieving a given predicate is over a threshold. At each step, the new intermediate predicate represents a *locally-minimal PAXp* corresponding to the state where the most important action took place for reaching the predicate computed at the previous step.

Although in the examples the actions identified as important 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 that for computational reasons, we study a single cause at each step). The user must be aware of this effect when computing B-HXPs.

B-HXP offers the user a credible alternative to forward HXP when studying the behaviour of an agent in long histories. Furthermore, this approach is agnostic with regard to the agent learning algorithm. As done in [20] for HXP, we also made the strong assumption for the use of B-HXP that the transition function is known. 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 study could 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 finding the 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. The various complexity results obtained indicate the inherent difficulty of the problem.

References

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

[2] D. Amir and O. Amir. HIGHLIGHTS: summarizing agent behavior to people. In *AAMAS*, pages 1168–1176. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 2018.

[3] L. Chen, P. Chen, and Z. Lin. Artificial intelligence in education: A review. *IEEE Access*, 8:75264–75278, 2020.

[4] J. A. Clouse. *On integrating apprentice learning and reinforcement learning*. PhD thesis, 1996.

[5] M. C. Cooper and J. Marques-Silva. Tractability of explaining classifier decisions. *Artif. Intell.*, 316:103841, 2023.

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

[7] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[8] N. C. 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] S. Greydanus, A. Koul, J. Dodge, and A. Fern. Visualizing and understanding Atari agents. In *ICML*, pages 1787–1796. PMLR, 2018.

[11] W. Guo, X. Wu, U. Khan, and X. Xing. EDGE: explaining deep reinforcement learning policies. In *NeurIPS*, pages 12222–12236, 2021.

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

[13] Y. Izza, X. Huang, A. Ignatiev, N. Narodytska, M. C. Cooper, and J. Marques-Silva. On computing probabilistic abductive explanations. *Int. J. Approx. Reason.*, 159:108939, 2023.

[14] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez. Explainable reinforcement learning via reward decomposition. In *IJCAI/ECAI workshop on explainable artificial intelligence*, page 7, 2019.

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

[16] S. Milani, N. Topin, M. Veloso, and F. Fang. A survey of explainable reinforcement learning. *CoRR*, abs/2202.08434, 2022.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[18] M. L. Olson, L. Neal, F. Li, and W. Wong. Counterfactual states for atari agents via generative deep learning. *CoRR*, abs/1909.12969, 2019.

[19] M. T. Ribeiro, S. Singh, and C. Guestrin. “Why should I trust you?": Explaining the predictions of any classifier. In *SIGKDD*, pages 1135–1144. ACM, 2016.

[20] L. Saulières, M. C. Cooper, and F. Dupin de Saint-Cyr. Predicate-based explanation of a Reinforcement Learning agent via action importance evaluation. In *ECML/PKDD workshop on Advances in Interpretable Machine Learning and Artificial Intelligence*, page to appear, Turin, Italy, 2023.

[21] L. Saulières, M. C. Cooper, and F. Dupin de Saint-Cyr. Reinforcement learning explained via reinforcement learning: Towards explainable policies through predictive explanation. In *ICAART 2023, Volume 2*, pages 35–44. SCITEPRESS, 2023.

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

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

[24] The White House OSTP. Blueprint for an AI Bill of Rights. Oct. 2022.

[25] S. Tsirtsis, A. De, and M. Rodriguez. Counterfactual explanations in sequential decision making under uncertainty. In *NeurIPS*, pages 30127–30139, 2021.

[26] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[27] J. van der Waa, J. van Diggelen, K. van den Bosch, and M. A. Neerincx. Contrastive explanations for reinforcement learning in terms of expected consequences. *CoRR*, abs/1807.08706, 2018.

[28] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

A Appendix

The first part of this section describes C4 and DC and the respective studied predicates. 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 in Figures 2,3 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 2 and 3, the importance scores given are w.r.t. either the initial predicate or the intermediate ones.

A.1 Description of the problems

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. An action corresponds to dropping a token in a column. The agent receives a reward only by reaching terminal states: 1, -1, 0.5 if the state represents an agent’s win, loss or draw respectively. As the agent does not know the next move of the opponent, transitions are stochastic. Five predicates were studied, including the obvious *win* and *lose*. For the other three, the initial state is compared with the final states of the generated scenarios. The predicate *control mid-column* is satisfied when the agent has more tokens in the middle column. The predicates *3 in a row* and *counter 3 in a row* are checked respectively when the agent obtains more alignments of 3 tokens on the board, and prevents the opponent from obtaining more alignments of 3 tokens on the board.

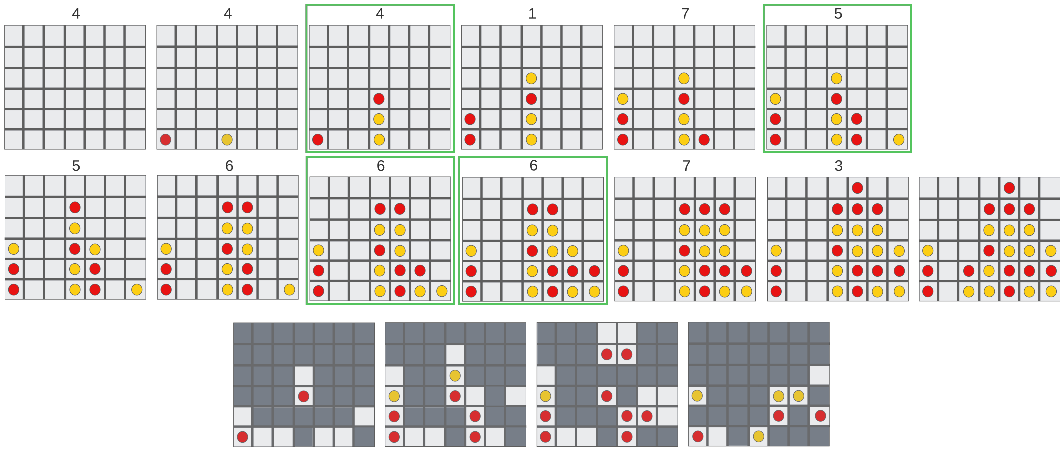

Figure 2: B-HXP for the *win* predicate in the C4 problem. Above: an input history of 13 states $s_0, \ldots, s_{12}$ and 12 moves (where a move is the choice of a column by the agent (yellow) to which the environment (red) responds). Below: the predicates found by B-HXP corresponding to the four important moves it finds, each highlighted by a green frame in the history.

Table 2: Importance scores in the C4 history of Figure 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

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 remain stationary. A drone's cover is the 3×3 square centered on it. A drone's cover is optimal when it does not contain any trees and there is no overlap with the covers of the other drones. Hence, its reward is based on its cover and neighbourhood. Moreover, it receives a negative reward in the case of a crash. 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. After an agent's action, the wind pushes the agent *left, down, right, up* according to the following distribution: $[0.1, 0.2, 0.4, 0.3]$ unless the action is *stop* or the agent and wind directions are opposite (in these cases the wind has no effect). 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. Local versions of predicates allow to check whether an agent reaches a perfect cover (*perfect cover*), gets a maximum reward (*maximum reward*), has no drones in its view range (*no drones*), did not crash (*crash*) and reaches a specific map region (*region*).

A.2 B-HXP examples

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 PAXpred predicates (see the line below the history in Figure 2), 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

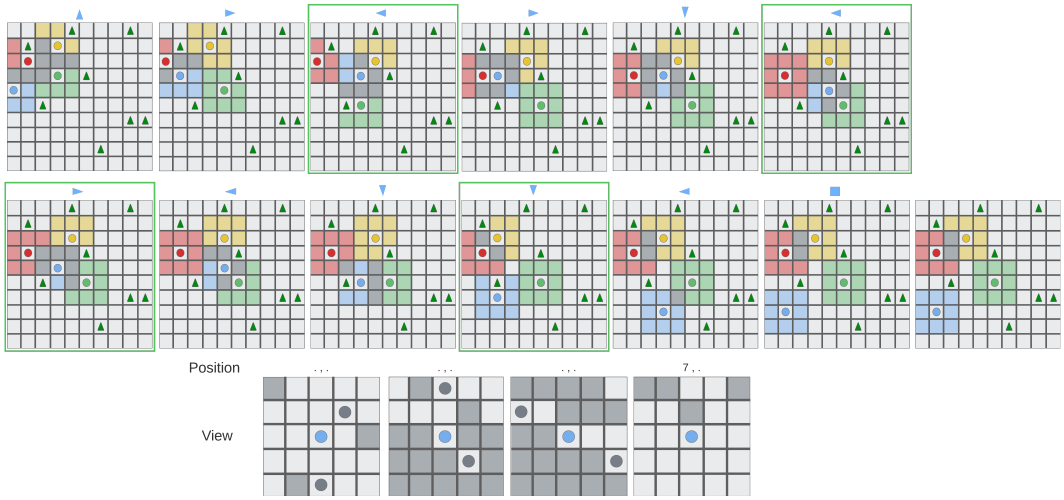

Figure 3: B-HXP for the *perfect cover* predicate in a DC problem history. The rightmost situation in the last row, corresponding to the first PAXpred predicate, states that the blue drone is in row 7 and that the light grey squares are free. The other intermediate predicates impose the relative positions of two other drones and that some squares are free. The drones are represented by dots and the trees by green triangles.

Table 3: Importance scores in the DC history 3

Predicate	Time-step / Importance score		
perfect cover	9	10	11
	0.114	0.063	0.056
$\text{PAXpred}_{\kappa}(s_9, 1.0)$	6	7	8
	0.008	0.006	0.002
$\text{PAXpred}_{\kappa}(s_6, 1.0)$	3	4	5
	0.053	-0.013	**0.09**
$\text{PAXpred}_{\kappa}(s_5, 1.0)$	2	3	4
	0.034	-0.036	0.023

least 80% chance of achieving a win in the final position. However, as in the FL B-HXP, 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). Nevertheless, 3 out of the 4 important actions returned set up the winning diagonal.

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. Importance scores are presented in Table 3. The most important actions reflect the blue drone's strategy of moving away from the drones and trees in its cover. The predicates give a good intuition of the type of state (position and $5{\times}5$ view) the agent is trying to reach.

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 at an horizon l that respect d. Conversely, a less generic predicate makes it more difficult to evaluate an action. A too specific predicate d can lead to insignificant importance scores (close to 0) for the respect of d. In several histories, notably C4 ones, the predicates generated are not generic enough, leading to less interesting explanations.