Noname manuscript No.
(will be inserted by the editor)

Heuristics, Answer Set Programming and Markov Decision Process for Solving a Set of Spatial Puzzles*

Thiago Freitas dos Santos · Paulo E. Santos · Leonardo Anjoletto Ferreira · Reinaldo A. C. Bianchi · Pedro Cabalar

the date of receipt and acceptance should be inserted later

Abstract Spatial puzzles composed of rigid objects, flexible strings and holes offer interesting challenges for reasoning about spatial entities that are common in the human daily-life's activities. This motivates the use of spatial puzzles as domains of study in this work. The goal of this paper is to investigate the automated solution of this kind of problems by extending an algorithm that combines Answer Set Programming (ASP) with Markov Decision Process (MDP) and Reinforcement Learning (RL), called oASP(MDP). This method is capable of constructing the set of domain states *online*, i.e., while the agent interacts with a changing environment. The aim of the extension proposed in this work is to add heuristics as a mechanism to accelerate the learning process, resulting in the main contribution of this paper: the Heuristic oASP(MDP) (HoASP(MDP)) algorithm. Experiments were performed on deterministic, non-deterministic and non-stationary versions of the puzzles. Results show that the proposed approach can considerably accelerate the learning process, outperforming other state-of-the-art methods.

Keywords Heuristic · Markov Decision Process · Answer Set Programming · Reinforcement Learning · Spatial Puzzles

Thiago Freitas dos Santos is sponsored by FAPESP-IBM Proc. 17/07833-9. Paulo E. Santos acknowledges support from FAPESP-IBM Proc. 2016/18792-9. Leonardo Anjoletto Ferreira acknowledges financial support from Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001 Reinaldo Bianchi acknowledges financial support from FAPESP Proc. 2016/21047-3.

Thiago Freitas dos Santos
Electric Engineering Department, Centro Universitário FEI, Brazil

Paulo E. Santos
College of Science and Engineering, Flinders University, Australia and
Electric Engineering Department, Centro Universitário FEI, Brazil

Leonardo Anjoletto Ferreira
Electric Engineering Department, Centro Universitário FEI, Brazil

Reinaldo A. C. Bianchi
Electric Engineering Department, Centro Universitário FEI, Brazil

Pedro Cabalar
Department of Computer Science, University of Corunna, Spain

1 Introduction

The capacity of learning actions and sequences of actions, from domain interactions, to solve complex tasks is an essential ability for any intelligent agent immersed in the physical world. This is particularly critical with respect to spatial domains containing rigid, as well as flexible (or holed) objects, whereby the actions and their effects are non-trivial. This is the main challenge considered in this work, namely, learning sequences of actions necessary to solve a given task from the interaction with physical objects. In this paper, the task of interest is finding solutions for a set of spatial puzzles composed of rigid objects, flexible strings and holes. Not only are these types of elements the composing parts of common human scenarios, but they are also of interest to application areas such as robot surgery and machine maintenance, in which objects with distinct (or contrasting) characteristics must be carefully manipulated in order to achieve a particular goal (that could be the removal of a tumor, or the repair of a broken mechanism).

Previous work has considered the automated solution of this kind of spatial puzzles from a logical perspective [3,4,29], where the actions and their effects were explicitly formalized, allowing the definition of a simple planning system capable of solving a number of such puzzles [3]. These earlier approaches, however, did not have a learning component and could not cope with more complex, stochastic or non-stationary versions of the puzzles (i.e., versions in which the effect of actions is non-deterministic – *stochastic*; or versions in which the domain rewards, transitions, actions and states may change in unexpected ways as an agent interacts with the environment – *non-stationary*). These challenges are addressed in the present work by considering the spatial puzzles as a Reinforcement Learning optimization problem, in which actions that lead to the solution are strongly rewarded, whereas unfruitful actions are severely punished. To this end, the present paper investigates a combination of Answer Set Programming (ASP) [9] and Markov Decision Process (MDP) [34] in a novel algorithm called oASP(MDP) (presented in Section 2.4), that is applied to find solutions to non-deterministic and non-stationary variations of two spatial puzzles, named Fisherman's Folly and Rope Ladder (Section 2.1). In this algorithm, ASP is used to represent the domain as an MDP, while Q-Learning is the Reinforcement Learning (RL) algorithm used to find an optimal policy[1] for this MDP [11,16,19,28,31]. The proposed combination of MDP, RL and ASP has the following main advantages: (1) it creates a table (s, a, s') of possible transitions between states, given an action, as the agent interacts in the world; (2) it learns a list of constraints on the action execution with respect to certain states (optimizing the execution); and (3) it facilitates a heuristic extension of the RL procedure, accelerating the action-selection procedure.

This paper extends our previous work (reported in [11,31]) with two main novel contributions: 1) the introduction of heuristics defining a new method capable of accelerating the learning procedure of the oASP(MDP) algorithm (Section 3); and 2) the automated solution of spatial puzzle domains considering non-deterministic actions and interactions in a non-stationary environment (where states and actions can change in unexpected ways as the agent interacts with the world).

Experiments (Section 4) and results (Section 5) show that our approach outperforms the non-heuristic version of the oASP(MDP) and a heuristic-accelerated version

[1] An optimal policy in this context is an assignment of an action to each state maximizing the return values.

of Q-Learning, showing the advantage of our proposed combination of MDP, RL, ASP and heuristics to solve challenging spatial problems.

A preliminary version of this paper is available as an arXiv preprint [32].

2 Background

This section presents a description of the base concepts used in this work. Starting with the description of the spatial puzzles explored, we then present a brief description of Answer Set Programming (ASP), used to represent the Markov Decision Process (MDP), and the Reinforcement Learning technique used to find an optimal solution for this MDP. Finally, the non-heuristic version of the oASP(MDP) algorithm is presented (its heuristic version, which is the novel contribution of the present paper, is described in Section 3).

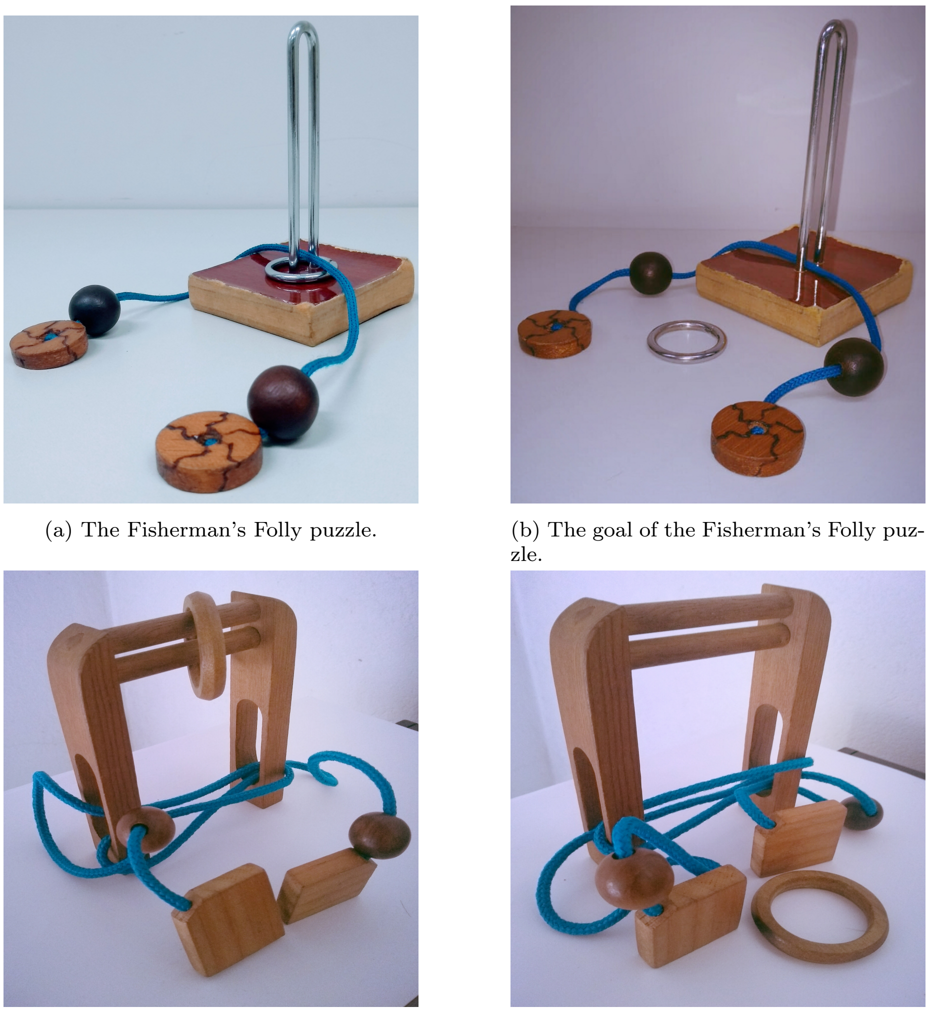
2.1 Domain

There is a large number of problem domains with the spatial characteristics we want to explore, ranging from tying shoelaces and simple sewing routines to complex topological knot untangling tasks applied to describe DNA replication [30]. The present work considers two problem domains containing flexible and holed objects: the Fisherman's Folly and the Rope Ladder puzzles; as well as their non-deterministic and non-stationary variants, first introduced in this paper (as we shall see in Section 4).

Both puzzles (Figure 1a and Figure 1c) are composed of the following elements: a Post with a Hole; a String; a Ring; a pair of Spheres, that are crossed by the String; a pair of Disks, fixed to the two tips of the String. The goal of these puzzles is to free the Ring from the entanglement of objects (Figure 1b and Figure 1d) by executing a sequence of actions. In this paper we consider the basic actions defined in [3], that is, passing an object or a tip of a long object through some hole in a given direction. An example can be the operation: "*pass the Disk through the Post Hole from left to right*". These passing operations are assumed to be complete: intermediate states in which the object is still partially crossing the hole are disregarded. This assumption was later removed in [4], but keeping it is still more convenient for finding the solution, since it avoids the explosion of irrelevant states.

In the Fisherman's Folly puzzle, the String crosses the Post in such a way that each Sphere and each Disk stays at one side of the Post. Besides, the Spheres can slide along the String, in contrast to the Disks that are fixed to the String's tips. The Spheres are larger than the Post Hole, so they cannot pass through the Post Hole without breaking the puzzle. However, the Spheres can pass through the Ring Hole. The Disks can cross the Post Hole, but they cannot pass through the Ring Hole, whereas the Ring can cross the Post Hole. These interactions between the elements of the puzzle can lead to some complex situations, such as the possibility of winding the String through the Post Hole (or through the Ring Hole) several times, which increases the state space and, consequently, the time needed to find the solution (this will be explored in Section 5).

The Rope Ladder puzzle has two Holed Posts, and the String crosses these Holes in four different points (see Figure 1c). In this puzzle there is a similar set of possibilities of interaction as in the Fisherman's Folly, with the addition that in the Rope Ladder



(a) The Fisherman's Folly puzzle.

(b) The goal of the Fisherman's Folly puzzle.

(c) The Rope Ladder puzzle.

(d) The goal of the Rope Ladder puzzle.

Fig. 1: Initial states and the goal states of the Spatial Puzzles studied in this work.

puzzle it is possible to pass the Post through both faces of the Ring Hole, in this case, the Post is not fixed to a base (as is the case with the Fisherman's Folly).

With these elements in mind, it is possible to categorize the puzzle's objects following the work reported in [3] that proposes three different categories for the elements:

1: *Regular (or simple) Objects*: Disk1, Disk2, Base.

2: *Objects with Holes*: Ring, Sphere1, Sphere2, PostHole1, PostHole2.

3: *Long Objects*: String, Post1, Post2.

The spatial puzzles are represented and manipulated by the agent by means of the PROLOG program introduced in [3], that is used in the paper as an oracle (a simulation of the puzzle) such that, given an action executed in a particular state of the puzzle, the oracle returns the description of the resulting state.

2.2 Answer Set Programming (ASP)

Answer Set Programming (ASP) [9] is a declarative logic programming language that facilitates non-monotonic reasoning. It has been successfully used to solve NP-Complete problems, such as the Traveler Salesman Problem, and it is designed to model and solve problems that deal with commonsense reasoning, such as spatial puzzles [31] or spatial non-monotonic reasoning [36].

According to Eiter et al. [9], ASP presents some advantages that justify its use, such as: the possibility to define which solutions are more desirable than others using a quality criterion, making it suitable for domains involving preference manipulation; the capability to work with missing information; and the so-called *choice rules*, which in practice, allows for the mapping between one input and several outputs. One important feature of ASP is that it relies on a declarative semantics based on the definition of *stable models* [20]. Gelfond and Lifschitz [13] define a stable model as "a possible set of beliefs the agent has, taking into consideration the premises of the program".

Lifschitz [20] defines an ASP program as a set of rules of the form:

$$\texttt{A :- L1, L2,..., Ln.} \tag{1}$$

where `A` is an atom, the rule *head*, and `L1,...,Ln` is the rule *body* consisting of literals, that is, either atoms `B` or their default negation `not B`. The symbol "`:-`" can be read as "if" and represents a (backwards) logical implication so that (1) stands for $\texttt{L1} \wedge \cdots \wedge \texttt{Ln} \rightarrow \texttt{A}$. The algorithm applied in this paper uses ASP *choice rules*, whose effect is non-deterministic and allows one input to have several outputs [11]. More formally, given one state "s" and one action "a", it is possible to have the states "$s1$" and "$s2$" as possible outputs when the action "a" is executed in state "s". This rule can be represented in ASP with the following formula:

$$\texttt{1 \{}s1\texttt{, }s2\texttt{\} 1 :- }a\texttt{, }s\texttt{.} \tag{2}$$

Formula 2 can be read as: given that the action "a" was executed in the state "s", one and only one state (between "$s1$" and "$s2$") can be chosen as a consequence of executing this action.

The premise of this work is that for each state $s \in S$ (a set of possible states of a domain) there is an ASP program that describes the effects of executing an action $a \in A$ (a set of possible actions that can be executed) using choice rules. The answer sets from these ASP programs represent the possible next states.

ASP represents constraints as headless rules. This kind of formula is mainly used to remove possible solutions that violate some rules. In the context of this work, these constraints are related to the description of forbidden states (that cannot be physically achieved), forbidden actions (that cannot be physically executed) and forbidden state-action pairs (actions that cannot be physically executed in a specific state). ASP is used in this work to find a set of states S of a Markov Decision Process (MDP), and then, to find all answer sets for each state an agent has permission to visit: all transitions allowed for the state-action (s, a) pair. Also, since ASP can revise (or update) previously obtained knowledge, and describe the transition rules of the domain, it is suitable to represent an MDP in a non-stationary domain, as we shall see further in this paper.

2.3 Reinforcement Learning (RL)

The algorithm proposed in this work applied Reinforcement Learning (RL) to find optimal solutions to the chosen domains. RL is a machine learning method in which the learning process happens through the interaction between an agent and the environment. Sutton and Barto [34] define RL over Markov Decision Process (MDP) that can be expressed as the tuple $\langle S, A, T, R\rangle$, in which:

S is the set of possible states in the domain;
A is the set of actions that can be executed by the agent;
T is the transition function that defines the probability of reaching a successor state $s' \in S$ when the agent executes action $a \in A$ starting from state $s \in S$:

$$Transition\ \ (T) : S \times A \times S \mapsto [0, 1]$$

R is the reward function responsible for providing the reward to the agent, when the agent is in a state $s \in S$ and executes action $a \in A$ to move to successor state $s' \in S$:

$$Reward\ \ (R) : S \times A \times S \mapsto \mathbb{R}$$

Besides this, Sutton and Barto [34] point out that an MDP assumes a first-order Markov property, which states that any state s contains all the information needed by the agent to decide the next action a to be executed.

In RL there are two main entities, the agent and the environment. The agent is responsible for learning and making decisions (executing actions), and the place (or world) in which the agent executes these actions is part of the environment. This RL framework of interaction is defined as follows [34]:

1. First, the agent chooses an action $a \in A$ at an instant t to be executed in the environment in a state $s \in S$;
2. Then, the environment answers the agent with the next state $s' \in S$ at the next instant t_{+1} together with the corresponding reward (a numerical value that describes the value of the action executed).
3. Finally, the agent updates the value of executing this action based on the received reward, using a formula defined by the specific RL algorithm being used.

In a nutshell, an MDP is used to formalize decision making processes in which *a priori* information about the transition and reward functions does not need to be fully specified. In this work an optimal policy for the MDP is obtained from the model-free online off-policy algorithm called Q-Learning [34].

The Q-Learning algorithm can learn an optimal policy following the RL framework of interaction defined above. To learn this policy, the algorithm uses an action-value function $Q(s, a)$ that is updated by selecting the action that maximizes the future gains at the end of each interaction between agent and environment. This is accomplished by the update function shown in Formula (3):

$$\begin{aligned} Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot (r + \\ \gamma \cdot \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)), \end{aligned} \tag{3}$$

where α and γ represent the learning rate and the discount factor, respectively, which (in this work) can be constant or set to decrease as the number of learning episodes increase. These interactions happen several times, and it is proven that at the limit (after infinitely many interactions) Q-Learning obtains an optimal policy for the problem. Although Equation 3 uses the *max* function, Q-Learning as proposed by Watkins [37] and used in this paper, is an online RL method since the learning occurs while the agent interacts with the environment. As it is an iterative process, the *max* function is applied on the current values of Q, generating a new value at each visited state-action pair (Q^{new}).

Informally, the Q-learning algorithm is based on a function that provides the *quality* (Q) of state-action pairs ($Q : S \times A \to \mathbb{R}$). From a randomly initialized table $Q(s, a)$ (Q table), at each time t and state s_t an action a_t is defined Q. This action executes a transition from s_t to s_{t+1}, whose associated reward $r_t(s_t, a_t, s_{t+1})$ is observed. The updated value of Q for (s_t, a_t) ($Q^{new}(s_t, a_t)$) is calculated from the value obtained considering previous interactions with the environment (weighted average of the previous value) and the value associated to the action that maximizes future gains ($\max_{a \in A} Q(s_{t+1}, a)$). An episode of Q-Learning ends when a terminal state is reached, or a predetermined number of interactions is executed.

One important point to highlight is that at the beginning of the interaction the Q-Learning agent has no knowledge about which action to choose, the choice is random. This lack of initial knowledge can lead to a poor initial performance for the agent, so it is possible to conceive a method to accelerate this learning procedure by reusing previously obtained knowledge. That is where the work reported in [1] becomes relevant for this paper. Bianchi et al. [1] present a new approach to the Q-Learning process, the Heuristically Accelerated Q-Learning (HAQL), with the possibility of adding heuristics to guide the learning process when the agent has previous knowledge about the task. The heuristics guide the learning process by helping the agent to choose an action in the action-selection phase, this is the only difference between the HAQL and the traditional version of Q-Learning, considering that the choice selection is using the ϵ-greedy strategy (i.e., a strategy in which the agent has a probability (1 - ϵ) of choosing an action that maximizes the value of the policy and a probability (ϵ) of choosing a random action). This is represented in Equation 4 below:

$$\pi(s) = \begin{cases} argmax_{a \in A}\{Q(s,a) + \xi H(s,a)^{\beta}\}, & \text{if } q \leq (1 - \epsilon); \\ random(A), & \text{otherwise,} \end{cases} \tag{4}$$

where $H(s, a)$ is the heuristic function that guides the choice of action, ξ and β are constant parameters that control the influence of the heuristic function, q is a random value between 0 and 1, responsible for defining the exploitation/exploration trade-off in the Q-Learning algorithm, $random(A)$ chooses some random action $a \in A$.

One limitation of Reinforcement Learning methods such as Q-learning, that is also inherited by HAQL, is that the set of states and the set of actions should be given as input. In the next section, we present the oASP(MDP) algorithm, combining ASP with Reinforcement Learning, that does not share these limitations. This algorithm is extended in Section 3 to solve the spatial puzzles considered in this work.

2.4 Online Answer Set Programming for Markov Decision Process

Ferreira et al. [11] define the Online ASP for MDP (oASP(MDP)) as an algorithm that combines the concepts described previously. In this algorithm, ASP is used to describe the MDP while Q-Learning is used to find the optimal policy for this MDP.

In this context, each transition $t(s, a, s')$ in the MDP is represented as an ASP choice rule of the form:

$$\texttt{1 \{}s'\texttt{, }s2'\texttt{, }s3'\texttt{\} 1 :- }a\texttt{, }s\texttt{.} \tag{5}$$

In this way every new state transition discovered by the agent (e.g., s', $s2'$ and $s3'$) is added to the choice rule related to an action. This is obtained with respect to all actions and states visited by the agent in the domain [11]. So, having the description of possible transitions for each action and each state that has been visited as an ASP program, an ASP solver can be used to generate the set of states, actions and transitions. Therefore, the underlying MDP is approximated by the answer sets at each iteration of the algorithm in the following sense: if the ASP program contains a rule $1\{s1, s2\}1 : -a, s$, then the MDP has non-zero probability in the transitions $\langle s, a, s1\rangle$ and $\langle s, a, s2\rangle$, although the transition probability value is unknown to the agent. Finally, through the interactions between the Q-Learning agent and the environment, the agent can define the action-value function $Q(s, a)$ for each state-action pair (s, a).

```
1  Algorithm: oASP(MDP)
2  Input: The set of actions $A$, any action-value function approximation method $M$ (in
     this work, in particular, $M$ is Q-learning) and a number of episodes $n$.;
3  Output: The approximated $Q(s,a)$ function.
4  Initialize the set of observed states $S = \{\}$
5  while number of episodes performed is less than $n$ do
6      repeat
7          Observe the current state $s$
8          if $s \notin S$ then
9              Add $s$ to the set of states $S$.
10             Choose and execute a random action $a \in A$.
11             Observe the future state $s'$.
12             Update the (ASP) logic program for state $s$ by adding the observed
                 transition as a choice rule.
13             Find the answer set of the ASP program representing $(s,a)$
14             Update the description of $Q(s,a)$ with new state-action pairs that are not
                 restricted and the states that are achievable from $(s,a)$, given by the
                 answer sets of the ASP program related to $s$.
15         else
16             Choose an action $a \in A$ as defined by $M$.
17             Execute the chosen action $a$.
18             Observe the future state $s'$.
19         end
20         Update the value of $Q(s,a)$ as defined by $M$.
21         Update the current state $s \leftarrow s'$.
22     until the end of the episode
23 end
```

Algorithm 1: The oASP(MDP) Algorithm [11].

Algorithm 1 presents the pseudo-code for oASP(MDP). First, the algorithm receives three different parameters (line 2, algorithm 1) as input: the set of possible actions A the agent can execute in the environment; the RL method M (such as Q-learning, SARSA, MCMC etc.) that is used to approximate the action-value function $Q(s, a)$; and the number of episodes n to be executed. It is worth mentioning that, although the set of actions A is fixed, the action definitions (and, therefore, their effects) may change as the agent interacts with the world, since oASP(MDP) allows for knowledge revision.

After the initialization, the set of observed states is assigned to an empty list (line 4, Algorithm 1), since this set is built in an online fashion while the agent interacts with the domain. Next, the algorithm starts a loop (lines 5-22). In each repetition of this loop, the oASP(MDP) observes the current state s, and then it can take two different actions (depending if the current state s is in the set of observed states S or not): if $s \notin S$ (line 8, Algorithm 1), then s is added to S and a random action from the set A is executed. As a result of this, the observed transition is added as a choice rule (line 12), that is used to construct the MDP as answer sets (line 13). To finalize this branch, the description of the action-value function Q(s,a) is updated (line 14).

On the other hand, if $s \in S$ (line 15, Algorithm 1), then the action is chosen by the RL method M, with no update of the choice rule describing the transition, since this description was already obtained when the state was first added to set S.

Finally, after these two conditions are met, the update of the values in the action-value function $Q(s, a)$ happens according to the RL method M. The current state s is also updated. Now, the current state is the successor state s' that was observed when the action was executed.

Regarding the changes in the set of actions A, if the action executed by the agent in the environment was one that is illegal to be executed (e.g., trying to pass Sphere1 through PostHole1), the environment informs this to the agent, that does not take this action into consideration.

The capability of oASP(MDP) to handle non-stationary domains resides essentially in lines 12, 13 and 14 of Algorithm 1. Since the underlying MDP is obtained as answer sets of the ASP programs representing the transitions, a change in the states and actions would just imply the generation of new answer sets. It is worth noting also that this change does not affect the entire MDP, but only a localized portion of the domain. In contrast to Q-Learning, that needs to re-initialize the obtained values each time a change happens in the environment, oASP(MDP) can still use and apply the previously learned knowledge, even when a change in the environment happens.

By combining ASP with RL, we do not alter how the underlying RL algorithm works but only which states it explores. As the set of state-action pairs is built in an online fashion, and represented as ASP programs (that can be revised), the proposed algorithm uses only a minimal (relevant) subset of all state-actions pairs in the RL step. In other words, oASP(MDP) has at least the same optimal behavior as the underlying RL method used. However, by creating the set of state-action pairs as the agent explores the domain, oASP(MDP) is in average more efficient than its underlying RL method.

3 Heuristic oASP(MDP): The HoASP(MDP) algorithm

This section presents the adaptation of the oASP(MDP) algorithm to work with heuristics: the HoASP(MDP) algorithm. Heuristics are taken into consideration at the action choice (line 16 of Algorithm 1), replacing the ϵ-greedy strategy by the heuristic strategy represented in Formula 4. In order to illustrate the use of heuristics, an example applying HoASP(MDP) algorithm when dealing with the Fisherman's Folly puzzle is presented next. HoASP(MDP) is initialized with the set of actions A, a learning method M and the number of episodes n. After that, the agent initializes the set of observed states S to empty, which is the starting point of the current episode. Next, the agent verifies if the initial state $s0$ (Figure 1a) is in the set S (line 6, Algorithm 1), since in the beginning of the learning procedure S is empty, the agent must execute a random action (line 10, Algorithm 1), for example, *pass Disk1 through PostHole1 from right to left*. Following the action execution, the agent goes to the successor state $s1$, whereas $s0$ is added to S and the agent receives the corresponding reward. In the case of the spatial puzzles considered in this work, as we shall see further in this paper, the reward function strongly rewards actions that lead to the goal, strongly punishes illegal actions and is (almost) neutral in the other cases. Now the agent has knowledge about a transition in the domain, that can be translated to a choice rule (line 12, Algorithm 1) in ASP as:

$$\texttt{1 \{}s1\texttt{\} 1 :- }a\texttt{(ExecutedAction), }s0.$$

Where, `ExecutedAction` is the action *pass Disk1 through PostHole1 from right to left* executed by the agent. Each state $s \in S$ has an ASP file with all these transition rules, that is updated each time a new transition in that state is detected. With these transition rules, ASP obtains all answer sets for that state, representing the possible next states. The action-value function $Q(s, a)$ is, then, updated with new state-action (s, a) pairs that were obtained as answer sets of the related ASP program.

The agent is now in $s1$, all the steps described above are repeated in the same way. However, the following situation can happen: assuming the randomly chosen action was to undo what was done in state $s0$, executing the action *pass Disk1 through PostHole1 from left to right* leads the agent to $s0$ again. Since the agent is back in $s0$, and $s0 \in S$, then a new iteration of HoASP(MDP) occurs. The agent now is going to use the learning method M (line 16, Algorithm 1) to choose the action to be executed, and it is also at this instant that the heuristic guiding process takes place. Given a mapping between the state in the heuristic task and the original task, the agent chooses an action to be executed using Equation (4).

In the end of each episode, the received reward is used to update the values of the action-value function $Q(s, a)$, which, after several episodes, is the Q-Table that contains the optimal policy [34].

3.1 Heuristic oASP(MDP) to Solve Spatial Puzzles

After defining the general principles of the HoASP(MDP) problem solving method, this section describes how this method was adapted to solve the spatial puzzles (of increasing complexity) considered in this paper: simplified puzzles (relaxed versions of spatial puzzles), Non-Deterministic (spatial puzzles with non-determinism when an action is executed) and Non-Stationary (spatial puzzles where changes occur in the domain while the interaction happens).

In the deterministic and non-deterministic versions of the Fisherman's Folly puzzle, an admissible heuristic was obtained from a simplification of the puzzle solved using the original oASP(MDP). The Q-Table obtained in this case was used as heuristics to accelerate the learning process of the original (non-simplified) version of the puzzle. The Fisherman's Folly simplification kept the same configuration and relations of objects as in the original puzzle. The simplification was a constraint imposed on the quantity of string winding around the Post Hole. In this setting, the String cannot wind through the Post Hole more than twice.

The solution for a simplified of the Rope Ladder puzzle was also used as heuristics, but there are differences to consider since a distinct strategy was applied to obtain these heuristics. In the simplified Rope Ladder, the String is initially crossing the two Post Holes only once, which is enough to simplify the puzzle and its optimal policy. Another point of adaptation was the introduction of a function that maps a state in the simplified Rope Ladder to a state in the original Rope Ladder. Since the set of actions on both puzzles were the same, this mapping function matches the actions that lead the agent to a certain state in the simplified puzzle to the actions that lead the agent to the corresponding state in the original puzzle. In other words, the agent takes into consideration that the simplified puzzle, used as heuristic, starts from the same state as the original puzzle. Thus, each state visited by the agent in the original puzzle, through the execution of any action a, is a corresponding state in the simplified puzzle by executing the same action a.

With respect to the actions' formalization in the algorithm, the tuple $\langle CE, HE, HF \rangle$ was used, where:

1. *CE* is a set of *Crossing Elements*, $ce \in CE$ is an element of the puzzle that is going to pass through a hole.
2. *HE* is a set of *Host Elements*, $he \in HE$ is an element of the puzzle that hosts a hole.
3. *HF* is a set of *Hole Faces*, $hf \in HF$ is a face of a hole *ce* towards which *he* is going to pass. There are two possible faces: positive (+) and negative (-).

Any action is the manipulation of a *CE* that passes through a *HE* toward a direction (*HF*). This representation is general enough to describe the actions for all puzzles (and configurations) considered in this work. Since an action is the combination of a *CE*, a *HE* and a *HF*, the agent cannot choose an action with the same element as *CE* and *HE*. It is worth mentioning that the agent does not have initial knowledge about which *CE* is able to be passed through a *HE* due to size or shape constraints: for instance, the agent does not know whether Sphere1 can pass through PostHole1, or any other restriction. The agent learns these constraints by means of the accumulated negative rewards it receives while interacting with the environment.

In the Fisherman's Folly there is a set of 20 actions over the following elements: 6 *CE* elements {Sphere1, Sphere2, Post, Disk1, Disk2, Ring}; 2 *HE* elements {PostHole1, Ring}; and 2 *HF* {Positive, Negative}.

The formalization of the actions for the Rope Ladder is very similar to the Fisherman's Folly: the only difference is in the crossing and host elements, that in this case correspond to *CE*={Sphere1, Sphere2, Post1, Post2, Disk1, Disk2, Ring}, *HE*= {PostHole1, PostHole2, Ring}, leading to 28 possible actions to be chosen by the agent.

Like the work described in [29], the states of the puzzles are represented with a list of crossings per each long object in the puzzle. So, in the Fisherman's Folly, we have a list for the objects being crossed by String, and another for the objects being crossed by Post. For example, for the initial state shown in Figure 1a, the lists of crossings can be represented as follows:

1. chain(String) = [+Sphere1, +Post, +Sphere2]
2. chain(Post) = [+Ring]

where each list element is some object name from *HE* preceded by a hole face sign (the "exit" of the crossing). For instance, the crossing +Ring in the Post list means that the Post is crossing the Ring towards the positive face of the Ring Hole.

We conjectured in [29] that the complexity of the kind of problems considered in this paper is related to the complexity of the unknotting problem in knot theory, which falls in the complexity class NP [15]. A formal assessment of the complexity of such spatial puzzles and their computational solutions are still open issues. In terms of the relative search space, considering that this work assumed no previous information about how the actions are applied on which objects, given that Fisherman's Folly has 20 actions over 8 distinct elements and the size of the shortest solution is 5 steps, and Rope Ladder has 28 actions over 10 elements and the size of the shortest solution is 12 steps, in the worst case the former has a search space of order $O(5^{(20\times8)})$ whereas the latter has a search space of order $O(12^{(28\times10)})$. In the most general case, where winding the string around holes indefinitely is allowed, the search space of both puzzles is infinite in the worst case.

4 Experiments

The goal of this section is to evaluate how distinct RL approaches work with variants of the two spatial puzzles considered in this work, Fisherman's Folly and Rope Ladder, under distinct configuration settings. Four RL algorithms were applied to the puzzles, as described in Sections 2 and 3. Q-Learning [34], the original oASP(MDP) [11], the Heuristically Accelerated Q-Learning (HAQL) [1] and the Heuristic oASP(MDP) (Ho-ASP(MDP)), proposed in this paper. For the comparison among these methods, some parameters were fixed for all four algorithms in all experiments as follows: the discount factor was set to 0.9; ϵ (trade-off between exploitation and exploration) was fixed to 0.1 until the 4000th episode, after that the value was decreased with a rate of 0.01 at each 250 episodes, down to the value of 0.03; the learning rate was fixed to 0.2; the heuristic control value was set to 0.25. The rewards were defined as: -100 to illegal actions (those actions that do not change the current state), 1000 to actions that lead to the goal state and -1 to each action performed by the agent that leads to a different state. The agent can execute 500 actions per episode and the total number of episodes per trial is 6000. To produce statistically relevant results, 30 trials were executed for each experiment.

This paper considers the following distinct configurations of the puzzles:

- ***Simplified Fisherman's Folly:*** In this configuration, the elements of the puzzle are the same as in the original Fisherman's Folly (Figure 1a), but the agent cannot wind the String through the Post Hole (this is the only element with this restriction), simplifying the state space. Two algorithms were applied in this domain, the original oASP(MDP) and Q-Learning.
- ***Original Fisherman's Folly:*** This is the original Fisherman's Folly puzzle, presented in Figure 1a. All four RL algorithms assumed in this article were applied to this domain (cf. Sections 2 and 3). Regarding the algorithms that use heuristics, the Q-Table of the previous puzzle (Simplified Fisherman's Folly) was used to guide the learning procedure. This task extends the work presented in [31] with the application of oASP(MDP) to Fisherman's Folly with different configurations, exploring a distinct set of RL parameters.
- ***Non-Deterministic Fisherman's Folly:*** In this version, when an agent chooses an action to execute, there is an 80% chance of that action achieving the expected effect, a 10% chance of the action to be executed, but mistakenly aiming at the opposite hole face and a 10% chance that the agent executes no action (staying in the same state). All four RL algorithms described in this article were applied

to this domain. As in the original Fisherman's Folly, the solution to the simplified Fisherman's Folly was used as heuristics in the non-deterministic case.

- ***Non-Stationary Disk Fisherman's Folly:*** Although the work in [10] presents the application of the offline ASP(MDP) to Non-Stationary domains, in the present paper, we wanted to evaluate how the oASP(MDP) algorithm deals with environment changes when a previous learned policy can be a drawback for a new task. In this domain, only the original oASP(MDP) and traditional Q-Learning algorithms were executed, without heuristics. In the beginning, the domain allows for both Disk1 and Disk2 to cross through the Ring, but not through the Post Hole. The change happens after the 2000th episode, when the puzzle becomes the original Fisherman's Folly (where the Disks cannot pass through the Ring).
- ***Original Rope Ladder:*** This is the most complex puzzle in the set, since there are more elements connected in a more challenging configuration when compared to the Fisherman's Folly puzzle. The optimal sequence of steps to solve the Rope Ladder has 12 actions. All four RL algorithms described in this article were applied in this domain. Regarding the algorithms that use heuristics, the Q-Table of a simplified version of the Rope Ladder was used to guide the learning procedure. In the simplified version, there is only one crossing between the String and PostHole1 and PostHole2. Another constraint applied to both configurations of the Rope Ladder puzzle is that the agent cannot pass Disk1 through PostHole2 nor Disk2 through PostHole1 and it cannot wind the String through the Posts' Hole two (or more) consecutive times.

To evaluate and compare the algorithms considered in this work, four metrics were used: the number of steps to solve the puzzle; the accumulated return values; the number of states visited by the agent; and the number of state-action pairs in the Q-Table obtained during the learning process. The results and discussions of these experiments are presented in the next section.

5 Results and Discussion

This section presents the results of applying the algorithms considered in this work to distinct configurations of spatial puzzles. This facilitates a comparison of Q-Learning and oASP(MDP) with respect to their heuristically accelerated versions.

5.1 Simplified Fisherman's Folly (SFF)

On the Simplified Fisherman's Folly (SFF) puzzle, only the algorithms that do not use heuristics were executed: oASP(MDP) and Q-Learning. Since, providing heuristics for this puzzle would trivialize its solution. Besides, it is the solution of SFF that is used as heuristics to the more complex domains.

Figure 2a (summarized in Table 1) and Figure 2b show the graphs for the Number of Steps to solve the puzzle and the accumulated return values, respectively. It is possible to see that the learning curves of both algorithms are similar, this is due to the fact that the learning algorithm used by oASP(MDP) is Q-Learning.

Figure 3a and Table 3 show the number of visited states. The oASP(MDP) agent visits more states because it can remove illegal actions, leading the agent to execute more valid actions with new states as outcome. On the other hand, Q-Learning keeps these illegal actions, which implies in a lower exploration of the environment.

Finally, Figure 3b and Table 5 present the number of state-action pairs contained in the Q-Table, showing that Q-Learning accumulated more pairs. The graph in Figure 3b shows a distinction with respect to the graph of visited states (Figure 3a) because all the actions are already in the Q-Table once the state is visited. Since the Q-Learning algorithm does not revise its knowledge about actions, it does not remove illegal actions from the Q-Table, it only assigns negative rewards to them.

5.2 Original Fisherman's Folly (OFF)

All 4 algorithms were executed on the Original Fisherman's Folly (OFF). Figure 4a (summarized in Table 1) shows the graph with the Number of Steps to solve the puzzle. We can also see that using the solution of a simplified version as heuristics to a more complex domain accelerates the learning process from the beginning, this is because the solution of the SFF puzzle is part of the solution of the OFF puzzle. The Student's T Test in Figure 4c shows that these two curves present differences that are statistically relevant.

Figure 4b represents the graph for the accumulated return values, which also shows a better performance for the heuristic algorithms compared to their non-heuristic versions.

Figure 5a and Table 3 show the number of states visited by the agent, oASP(MDP) visits more states than the other 3 algorithms. Comparing the number of interactions that is necessary to learn the optimal policy (Figure 4a and Table 1) and the number of visited states, it is possible to see that the possibility of winding the String through the Post Hole increases the complexity to solve the OFF puzzle, in contrast with the SFF puzzle (Figure 2a and Figure 3a). Considering the state space of the SFF puzzle, the oASP(MDP) agent visits around 6,000 states, whereas in the OFF puzzle the oASP(MDP) agent visits around 25,000 states. Heuristic algorithms explore a smaller portion of the State Space than their non-heuristic versions, this is because the heuristics are responsible for guiding the learning process from the beginning, indicating to the agent which actions to execute and, consequently, which states to visit, accelerating these algorithms to find solutions.

Finally, Figure 5b and Table 5 show the number of state-action pairs present in the Q-Table. Q-Learning is the algorithm with the highest number of state-action pairs.

5.3 Non-Deterministic Fisherman's Folly (NDFF)

The number of steps to solve the Non-Deterministic Fisherman's Folly (NDFF) is shown in Figure 6a and Table 1. It is possible to see that, at the beginning of the learning process, the heuristic algorithms perform better than their related non-heuristic versions. Student's T test in Figure 6c indicates that this difference is statistically relevant.

Figure 6b shows a graph with the accumulated return values, in which the heuristic algorithms present higher return values at the beginning of the process. Another point to consider is that oASP(MDP) has higher return values than Q-Learning in the beginning of the interactions, but as the number of interactions increases, this difference declines until they achieve the same level.

The graph in Figure 7a (summarized in Table 3) shows that oASP(MDP) explores more valid states than all the other algorithms, while Figure 7b and Table 5 show that Q-Learning has more state-action pairs in the Q-Table. Since, in this configuration, it is also possible to wind the String through the Post Hole, the oASP(MDP) agent explores a higher number of states than in any other configuration of this puzzle (Table 3), visiting around 80,000 different states. Thus, the algorithms take longer to learn (Table 1).

Although this is a non-deterministic problem, the algorithms were still able to use heuristics to accelerate the learning process, because the action choice procedure depends on the heuristics influence only at the moment the agent is actually choosing the action to be executed. Thus, even though these actions have non-deterministic effects, in the overall learning process, this non-determinism does not present a complication in terms of heuristics choice.

5.4 Non-Stationary Disk Fisherman's Folly (NSFF)

Considering the results related to the Non-Stationary Disk Fisherman's Folly (NSFF) puzzle, Figure 8a and Table 2 show that Q-Learning and oASP(MDP) algorithms have a similar performance until the change in the environment happens (at interaction 2000). After that, Q-Learning performs better than oASP(MDP), because the latter does not achieve the optimal policy. This is a statistically relevant difference as demonstrated by the Student's T Test in Figure 8c. The way the change happens in the environment impacts the performance of the oASP(MDP) agent since (unlike Q-Learning), in order to improve performance its Q-Table is not re-initialized, so the previously learned optimal policy negatively influences the agent's behavior in the new environment after the change. In our experiments, the oASP(MDP) agent did not receive enough negative reward in order to overturn the old policy, which led the agent to converge to a sub-optimal policy for the second part of the learning process.

Although there is a difference in the number of steps (Table 2) to solve the puzzle after the change occurs, this difference cannot be observed in the accumulated return values graph (Figure 8b) because the reward at each step is small (-1) when compared to the reward for achieving the goal (+1,000).

Regarding the behaviour related to the Visited States (Figure 9a and Table 4), the oASP(MDP) keeps the Q-Table, even though there is a change in the environment, while Q-Learning needs to re-initialize its values. Figure 9b and Table 6 show that Q-Learning has more state-action pairs, even though it re-initializes the Q-Table. The number of Visited States and state-action pairs just grow for the oASP(MDP) agent because the Q-Table is not re-initialized, in contrast with its Q-Learning agent.

5.5 Rope Ladder

Results related to the Number of Steps to solve the Rope Ladder puzzle are depicted in Figure 10a and Table 1, showing that heuristics greatly accelerate the learning process with respect to the non-heuristic algorithm. This difference is statistically relevant as shown by the Student's T Test (Figure 10c). Besides, the HoASP(MDP) also presents, for a brief number of interactions (around 300 interactions), a better performance than the Heuristically Accelerated Q-Learning, the Student's T Test supporting this difference is in Figure 11a.

Figure 10b presents the graph for the accumulated return values, it is worth noting how these values change over time: at the beginning of the interaction, the non-heuristic algorithms present higher return values than the heuristic algorithms, but this relation changes as the number of interactions increase, since the heuristic algorithms start to get higher return values.

One important observation that can be made about this domain is in relation to the accumulated return values graph (Figure 10b) and the number of visited states (Figure 11b). It is possible to see that at the beginning of the learning process (in the zoomed part of the graph), the algorithms that do not use heuristics have a higher return value than the algorithms that use heuristics. This happens because the number of visited states does not increase in the same proportion as the reward value for the non-heuristic algorithms. Since the non-heuristic algorithms visit the same states more often, they take longer to explore the state space of the domain, which leads to the execution of fewer illegal actions, as they have already received rewards indicating which actions are better in those known states. This is true only at the beginning of the interactions, after that, the heuristic algorithms explore the state space receiving higher rewards. The return values are higher in the heuristic algorithms, while the non-heuristic algorithms take longer to get to the same results.

Finally, the graph of the state-action pairs in Figure 11c and Table 6 show that Q-Learning explored a higher number of state-action pairs than the other algorithms.

The results presented in this section are summarized in the tables below, each of which representing one parameter used to evaluate the algorithms investigated in this work. Tables 1, 3 and 5 summarize the results for the stationary puzzles, while Tables 2, 4 and 6 summarize the results for the non-stationary puzzle. When a table has a "X" value in a cell, it means that the related algorithm was not executed. In the tables describing results for the non-stationary cases, the columns "oASP(MDP) Change" and "Q-Learning Change" refer to the results obtained after the change in the environment occurs.

Since the values for the "Number of Interactions" and the "Accumulated Return" convey similar conclusions, tables for the latter were omitted in this paper.

Considering the "Number of Steps" parameter for the stationary puzzles (Table 1), results show that both heuristically accelerated algorithms (HoASP(MDP) and HAQL) present better performance than their non-heuristic versions (oASP(MDP) and Q-Learning, respectively). In the non-stationary domain, Table 2 shows that oASP(MDP) and Q-Learning algorithms achieved a similar performance. However, this is true only during the interactions that happened before the change is detected in the environment, after that just Q-Learning finds the optimal policy, while oASP(MDP) converges to a sub-optimal one.

The "Number of Visited States" parameter is presented in Tables 3 and 4. For the stationary puzzles, while oASP(MDP) explores a larger portion of the state space, the HAQL algorithm is the one that explores it the least. In the non-stationary case, oASP(MDP) never re-initializes its Q-Table, thus keeping all states visited in past interactions, in contrast to Q-Learning whose re-initialiazation removes all states that were previously visited. This distinction led oASP(MDP) to visit more states after the change occurs (cf. Table 4).

Tables 5 and 6 summarize the values for the "Number of State/Action Pairs" parameter. Similarly to the "Number of Visited States" parameter, Tables 5 and 6 highlight the fact that, although the oASP(MDP) algorithms (original and heuristically accelerated versions) explore more of the state space, these algorithms do not present the highest numbers of State/Action pairs in the Q-Table. This is due to the fact that oASP(MDP) removes all illegal actions from the Q-Table, while Q-Learning keeps them.

Table 1: Approximated number of steps needed for the agent to start learning the optimal policy (and consequently the maximum reward) for the stationary puzzles.

Puzzle	Algorithms			
	oASP(MDP)	HoASP(MDP)	Q-Learning	HAQL
SFF	226	X	185	X
OFF	541	13	476	2
NDFF	2197	179	3388	91
Rope Ladder	1898	1022	2134	1242

Table 2: Approximated number of steps needed for the agent to start learning the optimal policy (and consequently the maximum reward) for the non-stationary puzzle.

Puzzle	Algorithms			
	oASP(MDP)	oASP(MDP) Change	Q-Learning	Q-Learning Change
NSFF	119	Optimal Policy not found	142	2213

Table 3: Number of visited states, during the interaction with the environment for the 6000 trials for the stationary puzzles.

Puzzle	Algorithms			
	oASP(MDP)	HoASP(MDP)	Q-Learning	HAQL
SFF	5895	X	4358	X
OFF	28379	4080	17117	1328
NDFF	79565	13455	61315	5705
Rope Ladder	15721	11967	14005	10235

Table 4: Number of visited states, during the interaction with the environment for the 6000 trials, for the non-stationary puzzle.

Puzzle	Algorithms			
	oASP(MDP)	oASP(MDP) Change	Q-Learning	Q-Learning Change
NSFF	4384	9710	3719	4366

The experiments were run on a 3.2GHz Intel Core I7-8700 with 16GB of RAM, in Ubuntu 16.04 LTS. The algorithms were implemented using Python 3.5, using ZeroMQ to provide message exchange between the environment and the agent. `SWI-Prolog` was used as the environment oracle and `clingo` was the ASP engine. The source

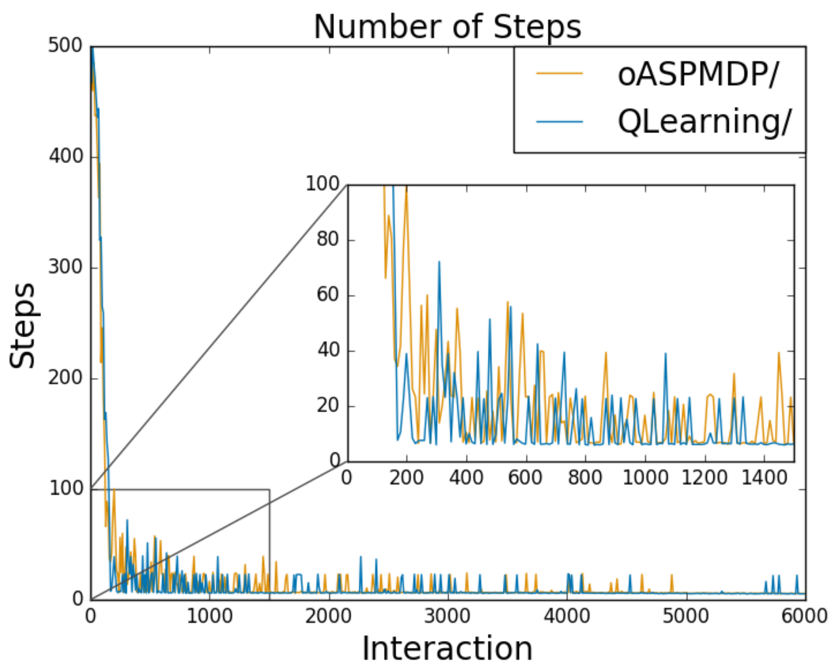
Table 5: Number of State/Action pairs present in the Q-Table for the stationary puzzles.

Puzzle	Algorithms			
	oASP(MDP)	HoASP(MDP)	Q-Learning	HAQL
SFF	39600	X	87157	X
OFF	253855	38570	342358	26573
NDFF	682128	126103	1226319	114118
Rope Ladder	92426	85005	392163	28660

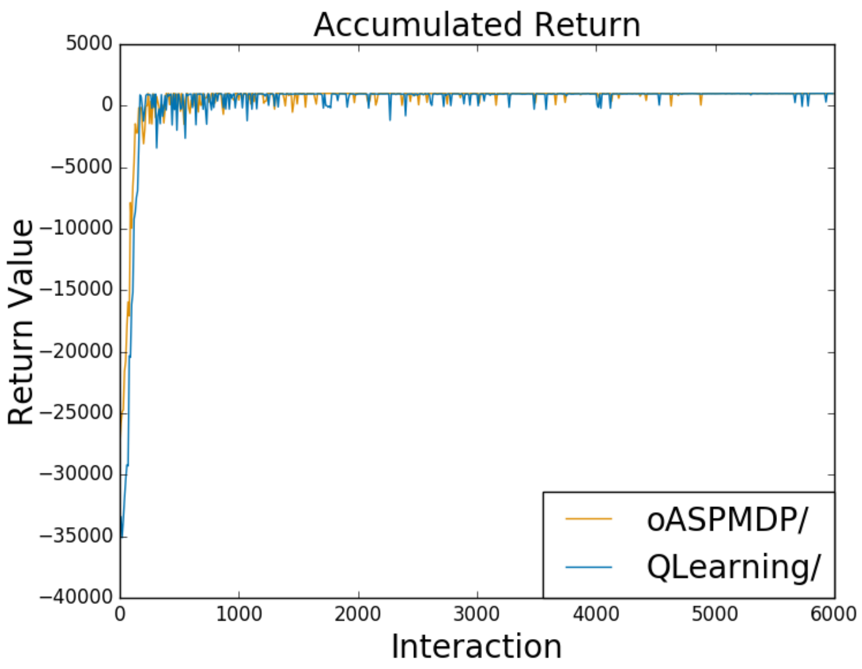
Table 6: Number of State/Action pairs present in the Q-Table for the non-stationary puzzles.

Puzzle	Algorithms			
	oASP(MDP)	oASP(MDP) Change	Q-Learning	Q-Learning Change
NSFF	36101	68303	59509	87322

code for the experiments is available at: `https://bitbucket.org/thiagomestrado/journalarticle/src`

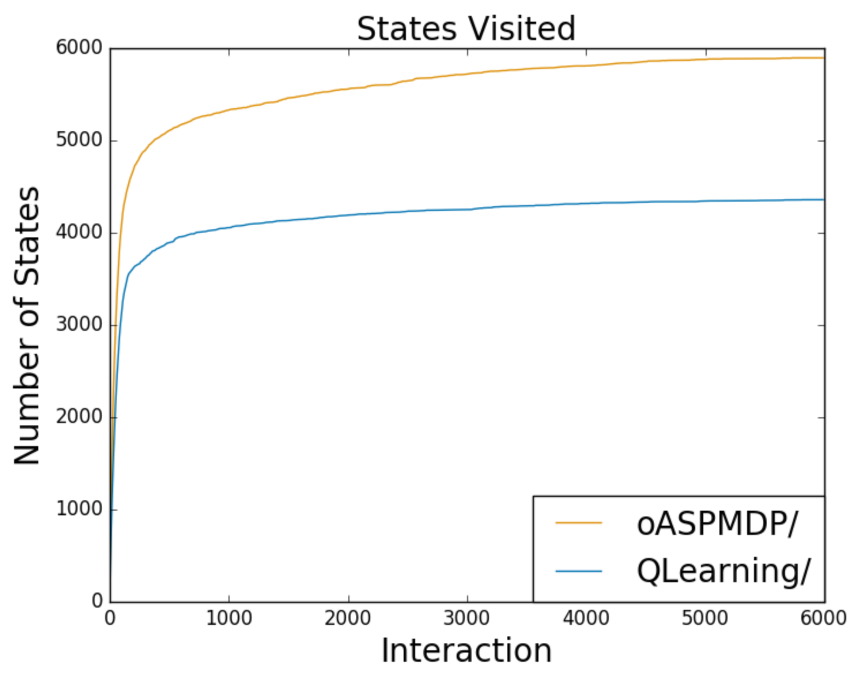


(a) Number of Steps to solve the puzzle.

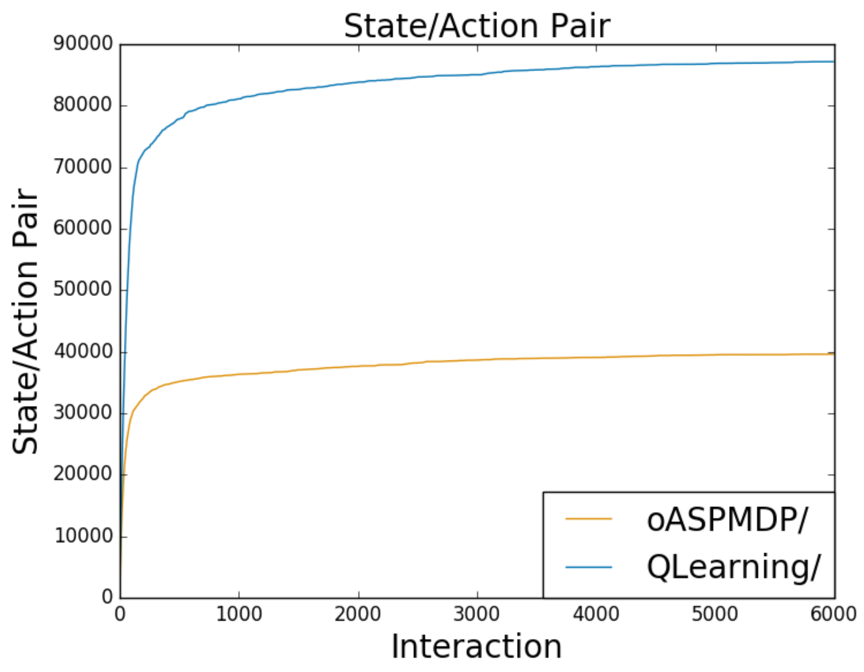


(b) Total accumulated Return received per episode.

Fig. 2: Number of Steps and Accumulated Return results for the Simplified Fisherman's Folly puzzle.

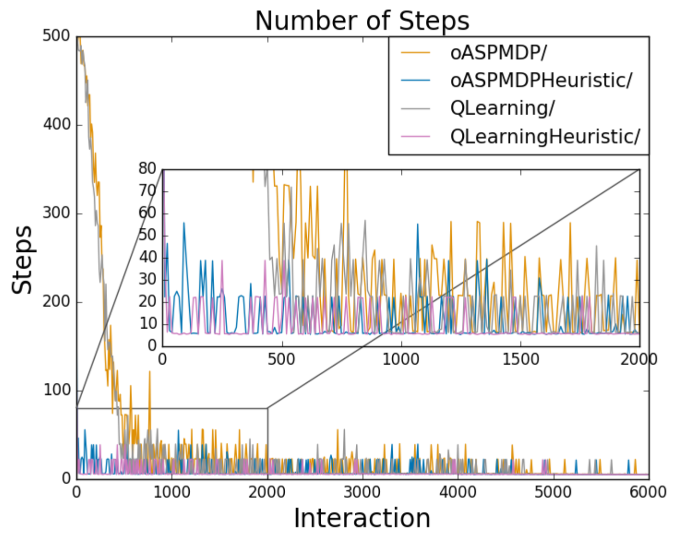


(a) Number of States visited by the Agent.

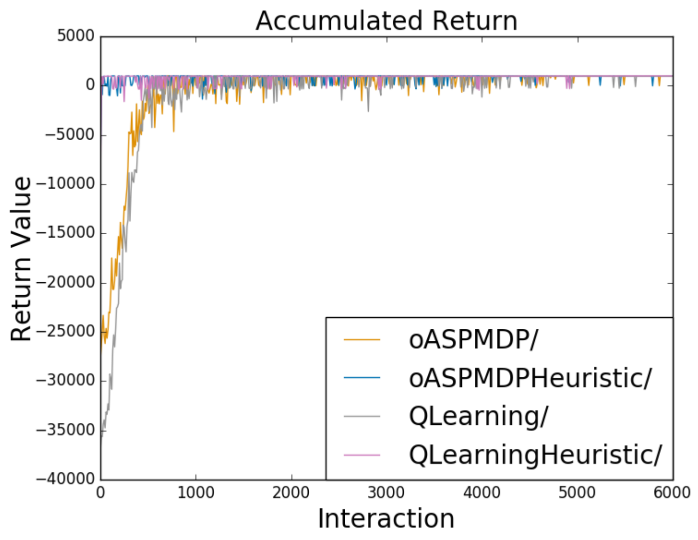


(b) Number of state-action pairs.

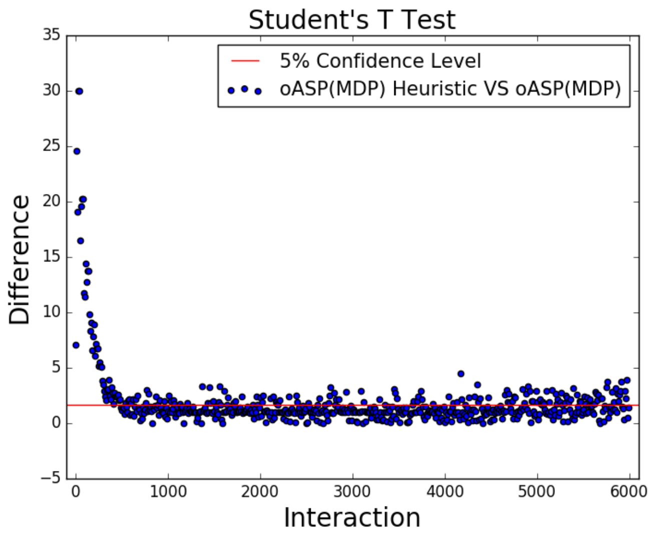
Fig. 3: States results for the Simplified Fisherman's Folly puzzle.



(a) Number of Steps to solve the puzzle.

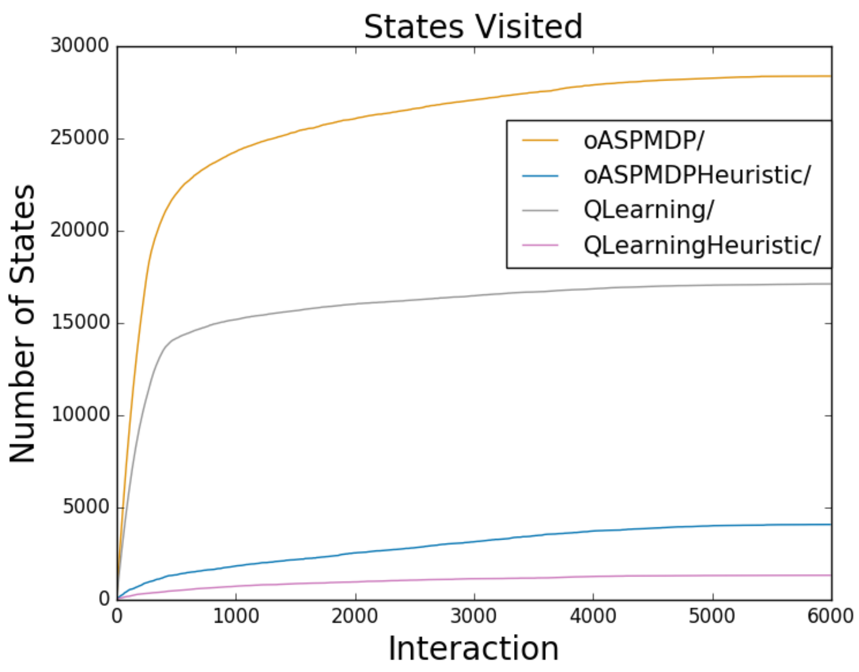


(b) Total accumulated Return received per episode.

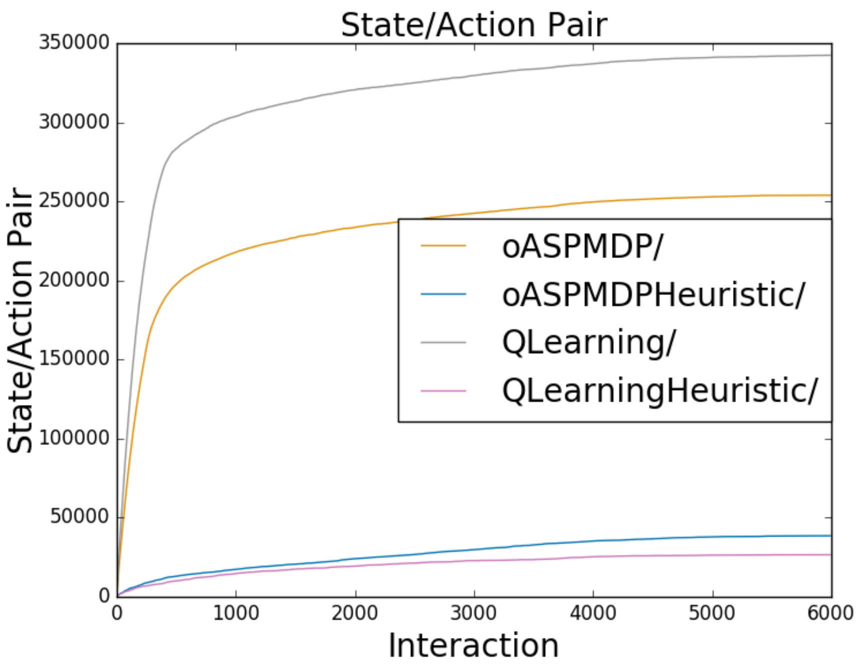


(c) T Test comparing the oASP(MDP) with Heuristic and the traditional oASP(MDP) .

Fig. 4: Number of Steps and Return results for the Original Fisherman's Folly puzzle.

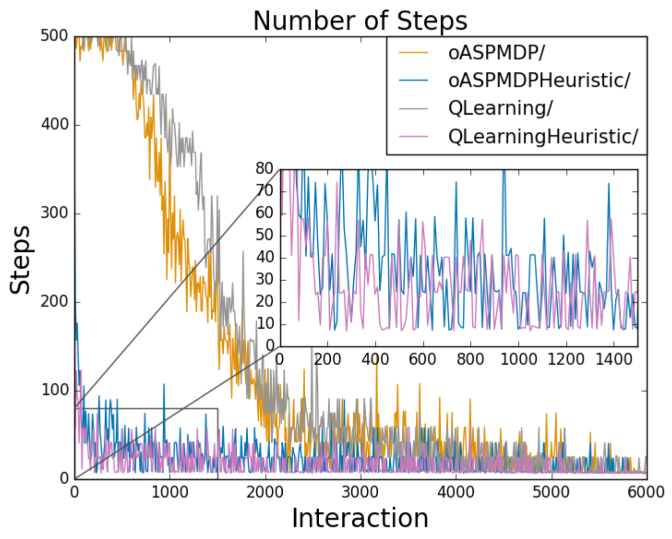


(a) Number of States visited by the Agent.

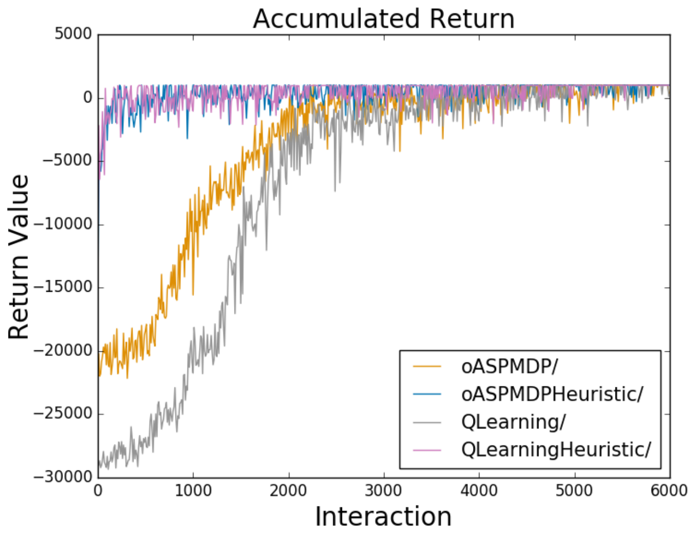


(b) Number of state-action pairs.

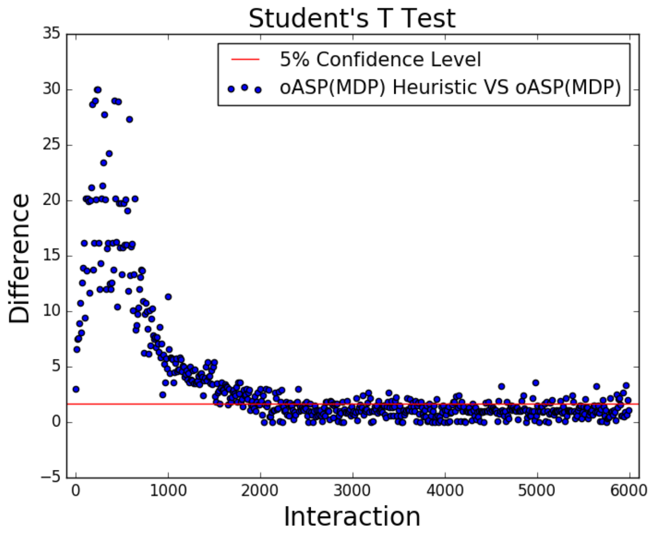
Fig. 5: States results for the Original Fisherman's Folly puzzle.



(a) Number of Steps to solve the puzzle.

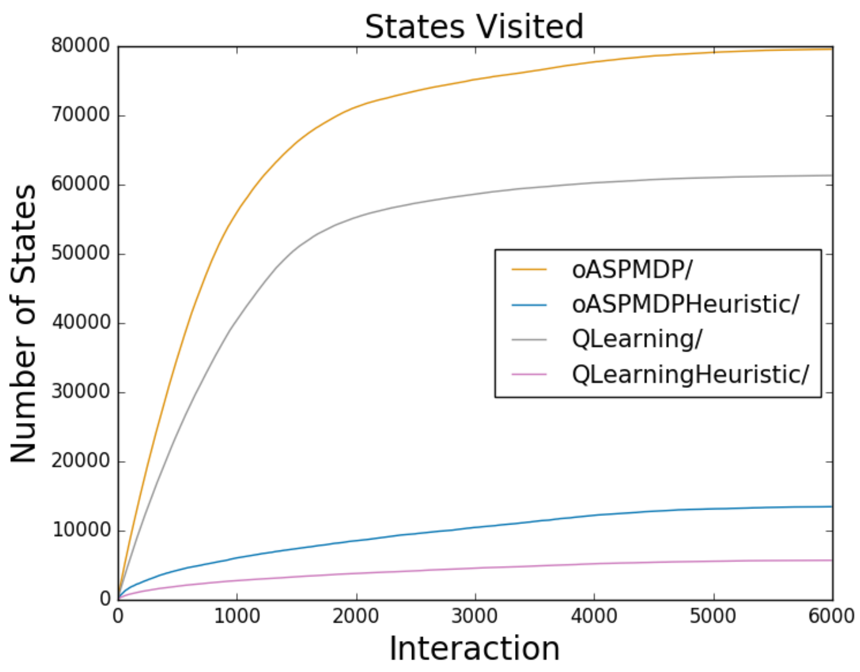


(b) Total accumulated Return received per episode.

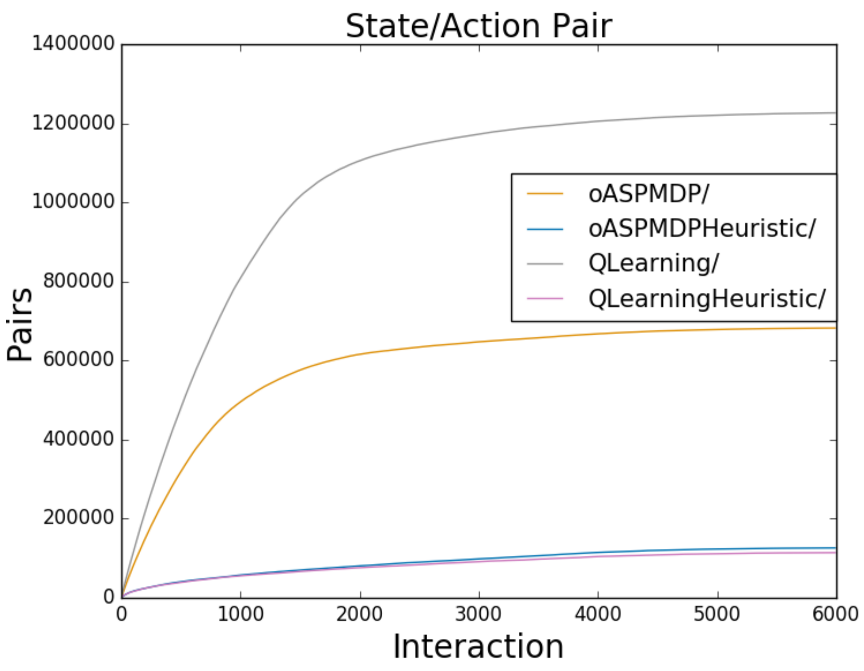


(c) T Test comparing the oASP(MDP) with Heuristic and the traditional oASP(MDP).

Fig. 6: Number of Steps and Return results for the Non-Deterministic Fisherman's Folly puzzle.

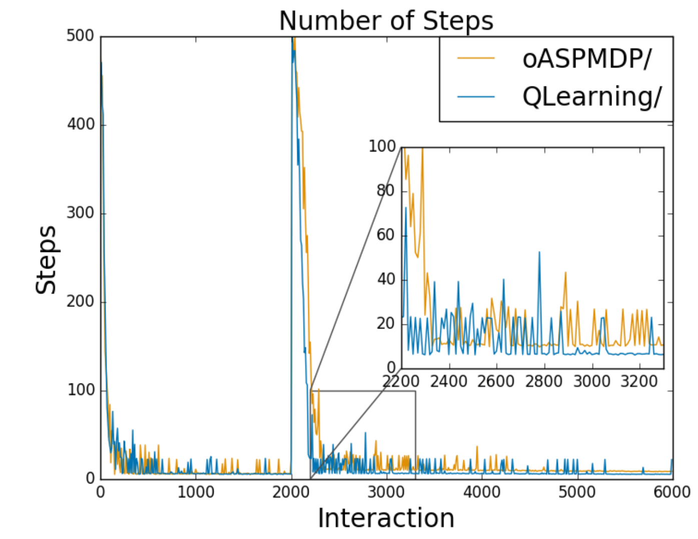


(a) Number of States visited by the Agent.

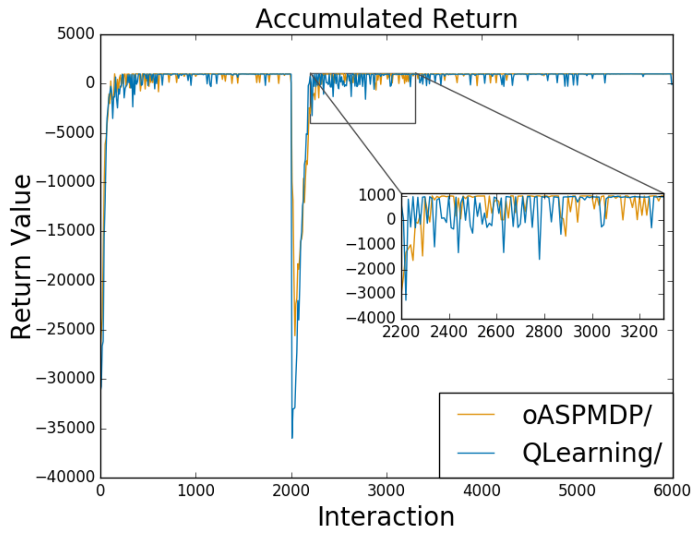


(b) Number of state-action pairs.

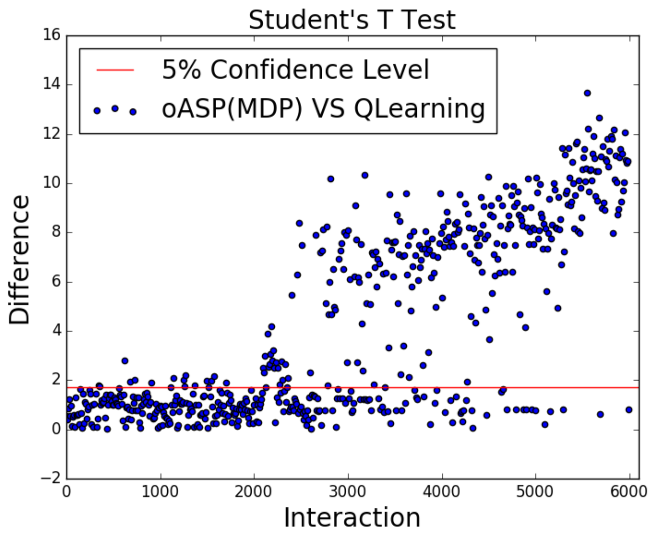
Fig. 7: States results for the Non-Deterministic Fisherman's Folly puzzle.



(a) Number of Steps to solve the puzzle.

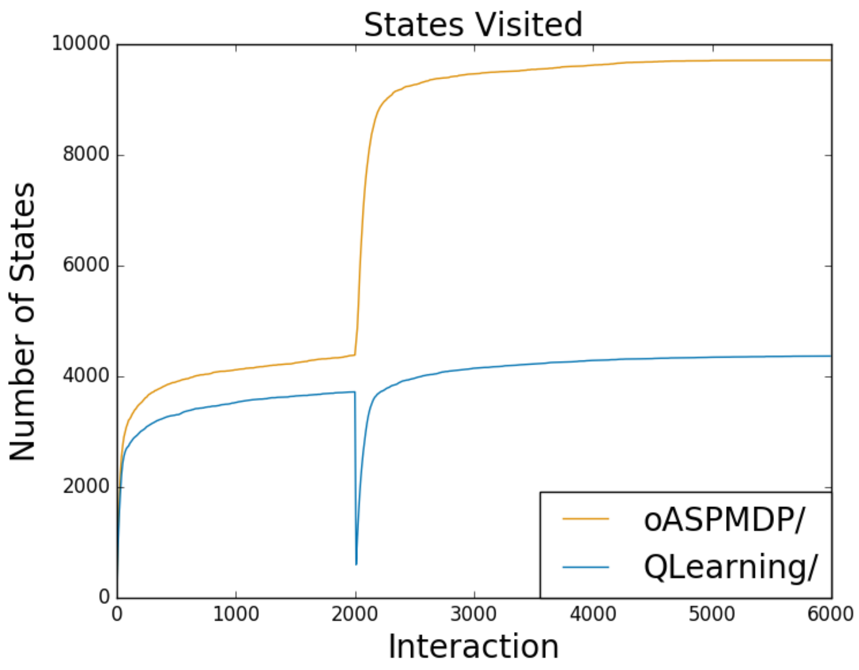


(b) Total accumulated Return received per episode.

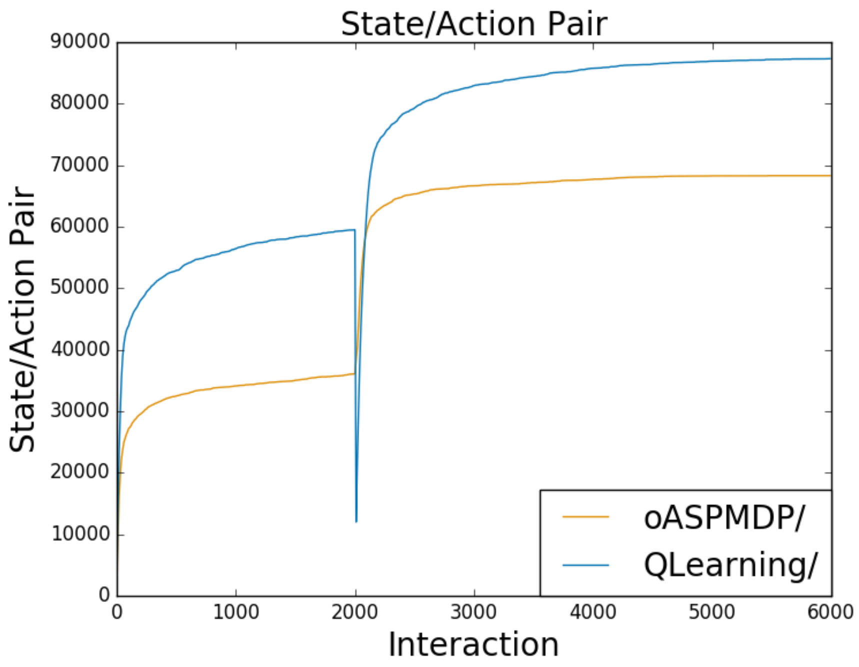


(c) T Test comparing traditional oASP(MDP) and the traditional Q-Learning.

Fig. 8: Number of Steps and Return results for the Non-Stationary Disk Fisherman's Folly puzzle.

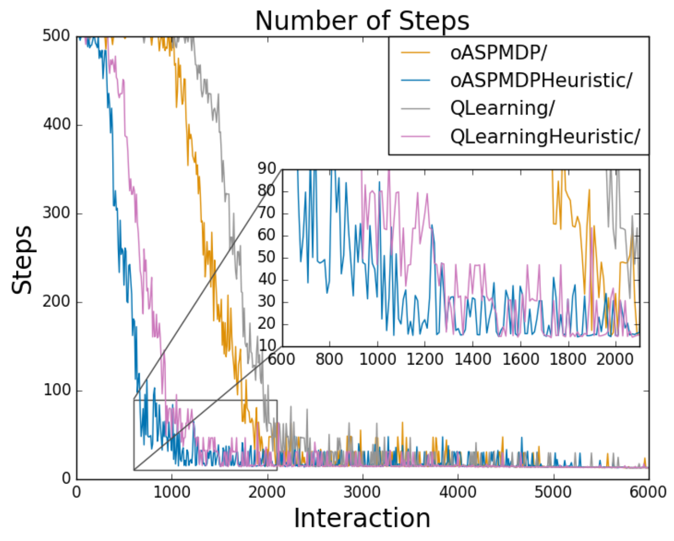


(a) Number of States visited by the Agent.

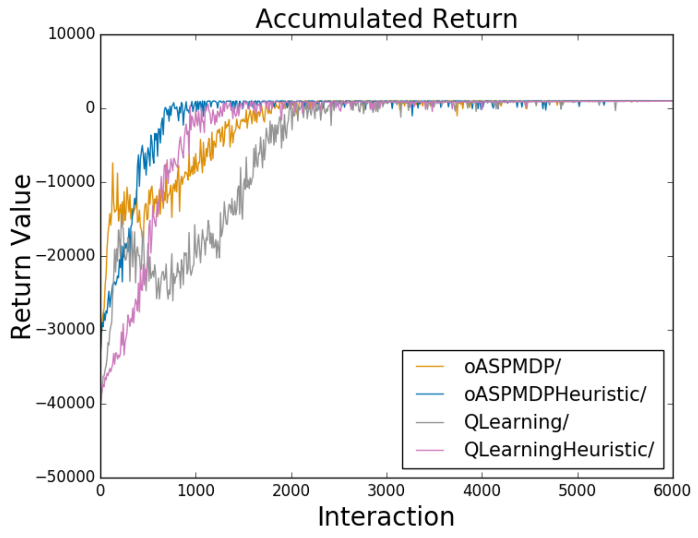


(b) Number of state-action pairs.

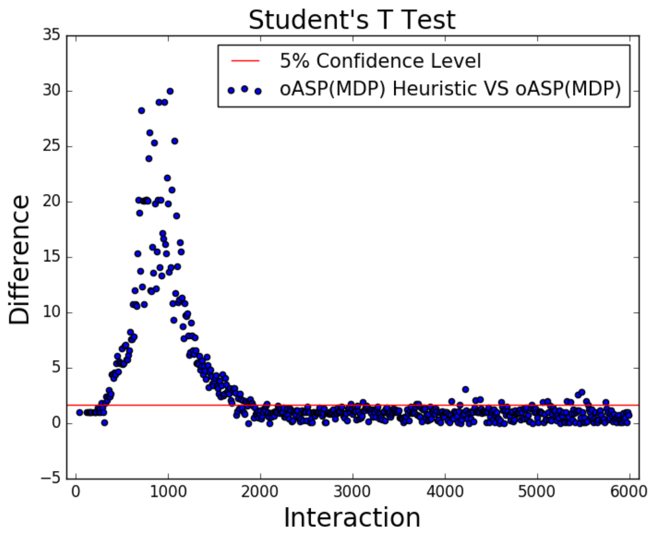
Fig. 9: States results for the Non-Stationary Disk Fisherman's Folly puzzle.



(a) Number of Steps to solve the puzzle.

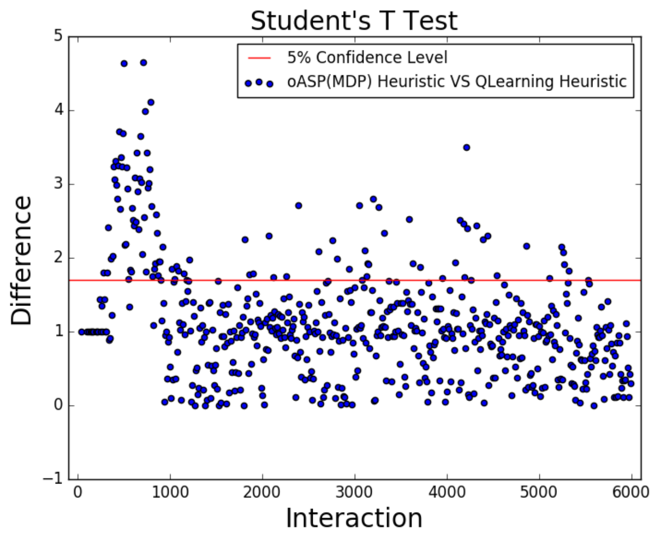


(b) Total accumulated Return received per episode.

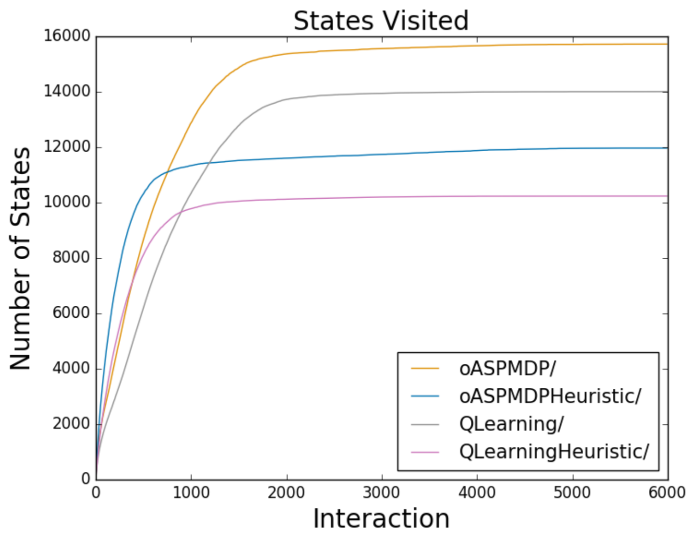


(c) T Test comparing the oASP(MDP) with Heuristic and the traditional oASP(MDP).

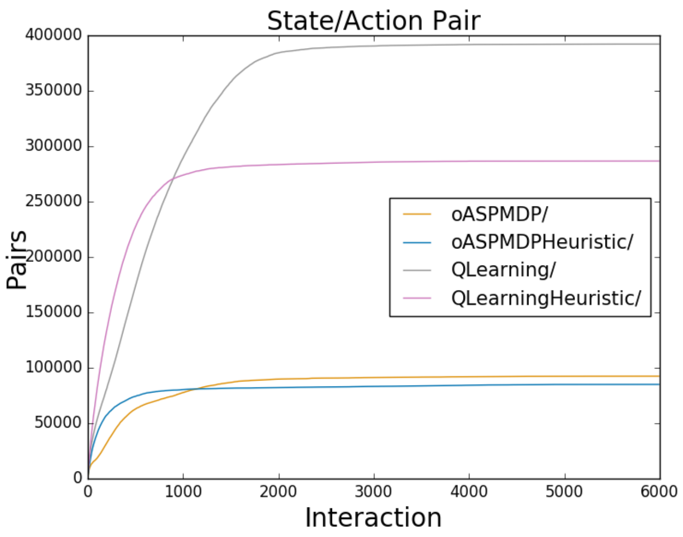
Fig. 10: Number of Steps and Return results for the Rope Ladder puzzle.



(a) T Test comparing the oASP(MDP) with Heuristic and HAQL.



(b) Number of States visited by the Agent.



(c) Number of state-action pairs.

Fig. 11: States results for the Rope Ladder puzzle.

6 Literature Review

This section presents related work to the research reported in this paper. Specifically, we cite literature on the combination of logic with MDPs and related to the use of heuristics in Reinforcement Learning. Additionally, considering non-stationary MDPs, we focus instead on the literature about changes with respect to state and action sets.

In order to find a solution to the Fisherman's Folly and Rope Ladder puzzles, the oASP(MDP) algorithm was used in this paper, an algorithm that combines logic programming and MDPs. One of the first works exploring a similar combination is [27] that describes a system integrating Relational Reinforcement Learning (RRL) with ASP and Event Calculus. In the context of [27], RRL can be implemented using variations of RL algorithms, e.g., SARSA and Q-Learning, to approximate the action-value function $Q(s, a)$, while ASP and Event Calculus are used to represent, compute and constrain the set of states and actions, and the learning task, providing the agent's background knowledge. As in our work, the results show that ASP can leverage the learning performance of RL based algorithms. In [27], the goal of the system is to solve the learning task while an agent interacts with deterministic and non-deterministic domains (the authors evaluate the system by using different deterministic and non-deterministic configurations of the Block's World). However, the system does not handle non-stationary environments, since it applies ASP after a determined action was chosen and the $Q(s, a)$ function was updated. In contrast, oASP(MDP) generates the answer sets before the action-selection phase and before the $Q(s, a)$ function updates, which makes our algorithm tolerant to changes in non-stationary settings. If on one hand RRL does not handle non-stationary domains, on the other hand the goal of RRL is to provide means for the agent to learn variations of the same problem (e.g., if it can solve the Block's World with 3 blocks, then it can reason about the optimal policy and solve a 5 blocks puzzle), which is a feature that oASP(MDP) lacks since it compares the description of states and actions to solve changes in the domain.

Garnelo et al. [12] present an architecture that combines Deep Reinforcement Learning (DRL) with Symbolic Reasoning (SR) separating it in two main blocks. The first is a Neural Network (NN), which gives as output a symbolic representation that is used as input to the second block of the architecture, responsible for the action choice process. In this context, Deep Learning (DL) is used to find a description of a set of states that can be described as rules for a probabilistic logic program. Similar to the work reported in the present paper, the inclusion of a reasoning component in the RL framework proposed in [12] leverages the learning task of the intelligent agent. However, where we use ASP for representing the set of state and actions, Garnelo et al. [12] use DL, thus, while the NN approach may be more robust than oASP(MDP) when considering small changes in the environment, it requires more processing time and computational power.

The work described in [18] explores the combination of ASP and RL in an algorithm called DARLING, which uses ASP for planning and reasoning, while RL is used to make the agent adaptive to changes in the environment. Since the RL learning process might require an unfeasible number of interactions between the agent and the environment to learn the optimal policy, DARLING uses planning to constrain the behavior of the agent, thus leading the agent to choose reasonable actions efficiently. As in oASP(MDP), DARLING can also be applied to changing environments, this can be achieved because both algorithms influence the learning task by adding a logical component that provides information (by constraining) which actions the agent should take. DARLING was tested on a service robot in an office-like environment, allowing the authors to show that, when DARLING is used, the robot can learn tasks faster, improving its performance over time. The difference between DARLING and oASP(MDP) is that while our approach does not need a preliminary phase (planning) to start learning and the answer sets are updated as the agent interacts with the environment, DARLING does not focus on finding the optimal solution of a problem and may be able to solve a problem faster than oASP(MDP), although in a suboptimal way.

Yang et al. [39] present a framework that integrates symbolic planning with hierarchical reinforcement learning (HRL), which deals with decision making processes in a non-deterministic domain. The idea is to use symbolic planning for option discovery in HRL, aiming at improving the learning and planning capabilities of an RL agent. In this case, the symbolic plans are used to guide the learning procedure. After this step, the learned experience is used again to further improve the provided plan for the HRL algorithm. Thus, the planning is constantly updated by the RL procedure. This approach is defined as Planning, Execution, Observation and Reinforcement Learning (PEORL), with the BC language being used to represent commonsense knowledge. The experiments were performed in traditional RL domains (the grid world and the taxi domain). There are two main results achieved in [39]: 1) the framework presents higher return values than traditional versions of RL and HRL; 2) and it is possible to obtain the optimal symbolic plan with PEORL, which is task not executed by oASP(MDP). This approach was applied only to deterministic domains, in contrast to the solution investigated in the present paper. Besides, we are focusing on constraining the actions by means of answer sets built incrementally as the interaction occurs, not by generating a plan. Using similar ideas, the work in [17] creates a specialized RL algorithm based on symbolic planning. Experiments show that this proposal can learn effective policies faster than traditional RL methods. The main goal of [17] is to use planning models and solutions calculated for them as guidance for solving RL tasks. To do that, the authors associate symbolic models to taskable RL environments. The HRL methods presented in [17] can at best converge to the hierarchically optimal policy, since the can (unintentionally) prune optimal policies. Hierarchies impose constraints over policies, but they allow for better description of the problem and a finer control of the agents' final behavior. Different from our approach that handles change in the state and action sets, taskable RL considers an environment with fixed dynamics, allowing for change only on goal conditions.

The work by Sridharan et al. [33] describes a combination of non-monotonic logics, using Answer Set Prolog, and probabilistic graphical models, with POMDP. This combination integrates an architecture named REBA that is useful to represent the domain knowledge and the agent's abilities and beliefs. POMDP is responsible for reasoning about the domain using data related to the agent's sensors and actuators, while the non-monotonic part of the architecture is used to provide a high-level description of the domain, that includes commonsense knowledge. In contrast to oASP(MDP), a REBA agent has initial knowledge about the task, allowing the use of ASP to create an abstract plan, which enables the agent to create and solve a POMDP for the given task. Although experiments were executed in a non-deterministic domain, there were no experiments performed in non-stationary settings. The authors take into consideration the history of a dynamic domain, which typically includes a record of actions executed and observations obtained (by the robot), which bears some similarity with the oASP(MDP), that uses the observations from the agent to build the answer sets responsible for constraining the action set. However, unlike oASP, REBA can not be applied to some of the domains used in this work. In contrast, REBA's algorithm has the ability to re-plan when the agent fails to achieve its goals, whereas oASP(MDP) needs to re-explore the search space in this case, loosing in efficiency.

The work reported in [40] combines Deep Q-Networks with symbolic representation of spatial relations between two objects. These relations make the Q-table an interactive relational model where the predicates representing the states are passed from the environment to the learning agent. The main difference with respect to the oASP(MDP) is in the state generation phase, since this phase in oASP(MDP) is the responsibility of the environment, with no use of an intermediate system that maps a set of predicates to a state. However, [40] uses a richer representation that is trained on a language corpus. Experiments were performed in the domain of mapping a person's intention to a goal.

Another related approach is presented in [43], where Answer Set Programming is combined with POMDP as a framework for a robot executing non-deterministic actions. In that work POMDP is used to describe actions and to deal with uncertainty about the robot's sensor readings. This is different from the algorithm investigated in the present paper mainly due to the base formalism used in the domain description. Regarding the process of providing information about similar tasks performed in comparable domains, the work in [43] also uses historical data to leverage this process, with the difference that we are using heuristics (extracted from the interactions with the environment but without historical data) and Zhang et al. [43] use hand-coded rules.

Also interested in the problem of combining a RL agent with explicit knowledge representation in order to avoid the execution of implausible actions, the work described in [26] explores the problem of adding commonsense knowledge to RL agents. The authors created a text-based game scenario to train and evaluate RL agents equipped with explicit knowledge about objects, their attributes and affordances. Results show that the use of commonsense knowledge implied a boost in performance as the agents explored the environment more efficiently. Analogous to the use of answer set programming in the present paper, the commonsense knowledge used in [26] could also prune the space state, thus reducing unfruitful and time-consuming exploration. The difference, with respect to the work described in the present paper, rests in the fact that Murugesan et al. [26] use manually annotated objects with qualifying properties for common sense, whereas the present work obtains a simpler description of the environment (responsible for pruning the state space and the set of actions) from the interaction between the agent and the environment without the use of common sense.

Problem solving in non-stationary domains is an important part of our study. To tackle the problem of obtaining an optimal solution to a non-stationary domain, the work in [28] presents an adaptation of a change-point algorithm that detects changes in the statistics of the environment, facilitating a RL agent to maximize the accumulative reward. Although the non-stationary rewards assumed in [28] are similar to the work presented here, the research reported in [28] does not consider changes in the domain states and actions, as well as in the effects of these actions.

A related approach to non-stationary MDPs, where change occurs in the set of states, is presented in [22] with the introduction of Continual Reinforcement Learning (CRL) methods. Lomonaco et al. [22] use Deep Reinforcement Learning (DRL) methods derived from A2C for a robotic agent to learn to navigate in a 3D maze. This maze can suffer changes on the light conditions, on the wall textures, as well as on the shapes and colors of objects or a combination of these. By measuring the average return, CRL can determine when the environment changes and adapts its Q-function representation to the new environment, without re-starting the Q-function learned in the previous configuration of the environment. Thus, the agent's description of the environment increases as the environment changes in a way that is similar to oASP(MDP). While, our use of answer sets makes the representation more flexible and compact, their approach is less prone to errors due to small changes in the environment.

In [23] the authors deal with the shortcomings of Reinforcement Learning and Knowledge Representation and Reasoning (KRR) by combining knowledge from humans with policies learned from a RL agent to dynamically compute task-specific planning models under unexplored new environments. Results show that there is an improvement in the model based RL, with experiments performed using a mobile robot working on dialog, navigation and delivery tasks. Like our work, Lu et al. [23] also represent the knowledge about the dynamics of the environment in a declarative form. In our case, since the set of state-action pairs is built in an online fashion and represented as ASP programs, our algorithm uses only the relevant subset of state-action pair to decide which action to take (in the RL step), but it is currently not expressive enough to represent the environment dynamics explicitly.

The oASP(MDP) algorithm is adapted in this paper to consider heuristics (in a transfer learning setting) following the ideas developed in our previous work. The Heuristically Accelerated Reinforcement Learning (HARL) is proposed in [1] that applies heuristics to accelerate and guide the reinforcement learning procedure. The authors show that good heuristics are very useful to guide the action-selection phase in RL, when an agent has little knowledge (especially at the beginning of the learning procedure). Moreover, as the interactions happen, the agent presents a better global performance due to the initial acceleration (which was also shown in other studies [6, 24,25,38]).

There are a variety of domains that can profit from transferring the knowledge learned in a source task to a target task. Morozs et al. [25] applies HARL to the domain of dynamic secondary spectrum-sharing in cellular systems[2]. As in our study, experiments showed that the use of heuristics can help to achieve high control of the sharing patterns in a totally autonomous way.

Another work describing the applicability of HARL is presented in [2], in which a class of algorithms that use Case-Based Reasoning (CBR) as heuristics in a transfer learning setting is studied. This approach was applied to two different robotic domains. First, the RL algorithm is applied to a source task (the task used to extract the heuristics); and after the learning stabilizes, the case base is built. Finally, the learned cases are transferred to be used in the target task. Similarly, the use of heuristics in our work is linked to a transfer learning process, since we obtain the heuristics by solving simpler versions of the domains, then reusing the knowledge in the new task (which is a more complex version of the puzzle). Zhang et al. [41] presents another related approach where the HARL algorithm is combined with a RBF Network. Experiments were performed in a grid world, showing that the use of the HARL approach improved the RL learning process.

HARL is used in [7] in order to find a pricing strategy that maximizes the total revenue in the commercialization of private data. The reported experiments show that

[2] A problem that deals with how to share the available spectrum, in the radio networks context, allowing for better voice calls and data transmissions, besides providing a good quality of service to the users.

the heuristics help to get higher returns in personal data transactions. Similarly, Liu et al. [21] present an application of HARL to the task of highway overtaking for an autonomous vehicle, where three heuristics were defined based on: 1)reward; 2)safety; and 3)efficiency. In contrast to our approach, that uses heuristic in the action choice step, in Liu et al. [21] the agents use the heuristic functions to decide the worst actions, that are ignored in the next iterations.

Although our approach focuses on heuristics as the basis for the transfer learning process, there are also other ways to handle this issue ([8, 35]). The work in [42] presents an approach to train the agent in a simple combat task (in this case, the context is for the Multi-Agent RL) and then to reuse this knowledge in a more complex task, a process that is similar to that proposed in the present paper; however, applied to multi-agent systems. These heuristics are also used to continuously train the model through self-play. Glatt et al. [14] propose an algorithm (called DECAF) to accelerate learning by building a library of cases that are reused in the moment of a new policy training. DECAF guides the training by dynamically selecting and blending policies according to their usefulness for the current target task, whereas in the work presented in this paper the state of the environment dictates the use of heuristics.

Focusing on target recognition underwater, the work in [5] presents an approach to transfer learning that includes a Neural Network (NN) to extract features of the task and then calculate the similarities between these features. The goal is to learn the feature data in the source task and then apply that to the target task, in order to reduce the repetition of similar data calculation in the reinforcement learning process. Instead of using a NN to map the task source to the target task, the research reported in the present paper use the components described by a logical program to transfer heuristics across tasks. Thus, although oASP(MDP) can learn a representation of the environment, the approach presented in [5] might be better at finding patterns from the environment observations, given its use of a NN.

7 Conclusion

This paper explored the use of heuristics in a method that combines Answer Set Programming with Reinforcement Learning in a Markov Decision Process (oASP(MDP)) applied on a set of spatial puzzles. This work considered two base puzzles composed of entanglements involving flexible strings, rigid objects and holes: the Fisherman's Folly and the Rope Ladder, where the latter has more objects submitted to more challenging relations than the former. Experiments were executed on distinct versions of these puzzles, defining deterministic, non-deterministic, and non-stationary domains. With these domains, the following four algorithms were compared in this paper: the traditional Q-Learning, the Heuristically Accelerated Q-Learning (HAQL), oASP(MDP) and the main contribution of this work: the heuristic version of oASP(MDP) (HoASP(MDP)).

Heuristics used in this work were obtained from the solution of relaxed versions of the domains considered. The results obtained show that these heuristics provided suitable information to guide and accelerate the learning process, where the heuristic-accelerated algorithms outperformed their relative non-heuristic versions with HoASP(MDP) showing the best performance overall. In particular, the oASP(MDP) and HoASP(MDP) algorithms demonstrated to be capable of exploring a larger portion of the valid state space by executing fewer illegal actions than their base learning algorithms (Q-Learning and HAQL, respectively). Last but not least, as HoASP(MDP) only needs the Q-table of a solved problem as heuristic, it can be used as a general-purpose problem solving method.

In order to explore further the use of ASP applied to Markov decision processes, future work will focus on using ASP to extract general domain rules (constraints, for example) and then apply these rules to reason about the domain, possibly guiding the learning process. In this way, we can have a more general framework for reusing previously learned knowledge while also exploring explicit descriptions of the domain states, allowing for readable (or explainable) outputs and elaboration tolerant solutions.

References

1. Bianchi, R.A., Ribeiro, C.H., Costa, A.H.: Accelerating autonomous learning by using heuristic selection of actions. Journal of Heuristics **14**(2), 135–168 (2008)
2. Bianchi, R.A., Santos, P.E., da Silva, I.J., Celiberto, L.A., de Mantaras, R.L.: Heuristically accelerated reinforcement learning by means of case-based reasoning and transfer learning. Journal of Intelligent & Robotic Systems **91**, 301–312 (2018)
3. Cabalar, P., Santos, P.E.: Formalising the fisherman's folly puzzle. Artificial Intelligence **175**(1), 346–377 (2011)
4. Cabalar, P., Santos, P.E.: A qualitative spatial representation of string loops as holes. In: Artificial Intelligence, vol. 238, pp. 1 – 10. Elsevier (2016)
5. Cai, L., Sun, Q., Xu, T., Ma, Y., Chen, Z.: Multi-auv collaborative target recognition based on transfer-reinforcement learning. IEEE Access **8**, 39273–39284 (2020)
6. Celiberto Jr, L.A., Matsuura, J.P., De Mantaras, R.L., Bianchi, R.A.: Using cases as heuristics in reinforcement learning: a transfer learning application. In: IJCAI Proceedings-International Joint Conference on Artificial Intelligence, vol. 22, p. 1211 (2011)
7. Chen, X., Chen, J., Chen, Y., Yang, J., Li, D.: Heuristic-q: A privacy data pricing method based on heuristic reinforcement learning. In: X. Sun, Z. Pan, E. Bertino (eds.) Artificial Intelligence and Security, pp. 553–565. Springer International Publishing, Cham (2019)
8. Da Silva, F.L., Costa, A.H.R.: A survey on transfer learning for multiagent reinforcement learning systems. Journal of Artificial Intelligence Research **64**, 645–703 (2019)
9. Eiter, T., Ianni, G., Krennwallner, T.: Answer set programming: A primer. In: Reasoning Web. Semantic Technologies for Information Systems, pp. 40–110. Springer, Berlin, Heidelberg (2009)
10. Ferreira, L.A., Bianchi, R.A., Santos, P.E., de Mantaras, R.L.: Answer set programming for non-stationary markov decision processes. Applied Intelligence **47**(4), 993–1007 (2017)
11. Ferreira, L.A., Bianchi, R.A.d.C., Santos, P.E., De Mantaras, R.L.: A method for the online construction of the set of states of a markov decision process using answer set programming. In: Recent Trends and Future Technology in Applied Intelligence, pp. 3–15. Springer International Publishing (2018)
12. Garnelo, M., Shanahan, M.: Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. Current Opinion in Behavioral Sciences **29**, 17–23 (2019). DOI 10.1016/j.cobeha.2018.12.010
13. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP, vol. 88, pp. 1070–1080 (1988)
14. Glatt, R., Da Silva, F.L., da Costa Bianchi, R.A., Costa, A.H.R.: Decaf: Deep case-based policy inference for knowledge transfer in reinforcement learning. Expert Systems with Applications **156**, 113420 (2020)
15. Hass, J., Lagarias, J.C., Pippenger, N.: The computational complexity of knot and link problems. J. ACM **46**(2), 185–211 (1999). DOI 10.1145/301970.301971. URL `https://doi.org/10.1145/301970.301971`
16. Homem, T.P.D., Santos, P.E., Reali Costa, A.H., da Costa Bianchi, R.A., Lopez de Mantaras, R.: Qualitative case-based reasoning and learning. Artificial Intelligence **283**, 103258 (2020)
17. Illanes, L., Yan, X., Icarte, R.T., McIlraith, S.A.: Symbolic plans as high-level instructions for reinforcement learning. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 30, pp. 540–550 (2020)
18. Leonetti, M., Iocchi, L., Stone, P.: A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. Artificial Intelligence **241**, 103–130 (2016)

19. Licks, G.P., Couto, J.C., de Fátima Miehe, P., De Paris, R., Ruiz, D.D., Meneguzzi, F.: Smartix: A database indexing agent based on reinforcement learning. Applied Intelligence pp. 1–14 (2020)
20. Lifschitz, V.: What is answer set programming?. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 8, pp. 1594–1597. MIT Press (2008)
21. Liu, T., Huang, B., Deng, Z., Wang, H., Tang, X., Wang, X., Cao, D.: Heuristics-oriented overtaking decision making for autonomous vehicles using reinforcement learning. IET Electrical Systems in Transportation (2020)
22. Lomonaco, V., Desai, K., Culurciello, E., Maltoni, D.: Continual reinforcement learning in 3D non-stationary environments. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops (2020)
23. Lu, K., Zhang, S., Stone, P., Chen, X.: Robot representation and reasoning with knowledge from reinforcement learning. arXiv preprint arXiv:1809.11074 (2018)
24. Martins, M.F., Bianchi, R.A.: Heuristically-accelerated reinforcement learning: A comparative analysis of performance. In: Conference Towards Autonomous Robotic Systems, pp. 15–27. Springer Berlin Heidelberg (2014)
25. Morozs, N., Clarke, T., Grace, D.: Heuristically accelerated reinforcement learning for dynamic secondary spectrum sharing. IEEE Access **3**, 2771–2783 (2015)
26. Murugesan, K., Atzeni, M., Kapanipathi, P., Shukla, P., Kumaravel, S., Tesauro, G., Talamadupula, K., Sachan, M., Campbell, M.: Text-based rl agents with commonsense knowledge: New challenges, environments and baselines. arXiv preprint arXiv:2010.03790 (2020)
27. Nickles, M.: Integrating relational reinforcement learning with reasoning about actions and change. In: International Conference on Inductive Logic Programming, pp. 255–269. Springer (2011)
28. Padakandla, S., Prabuchandran, K., Bhatnagar, S.: Reinforcement learning algorithm for non-stationary environments. Applied Intelligence **50**(11), 3590–3606 (2020)
29. Santos, P.E., Cabalar, P.: Framing holes within a loop hierarchy. Spatial Cognition & Computation **16**(1), 54–95 (2016)
30. Santos, P.E., Cabalar, P., Casati, R.: The knowledge of knots: an interdisciplinary literature review. Spatial Cognition & Computation **19**(4), 334–358 (2019)
31. dos Santos, T.F., Santos, P., Ferreira, L., Bianchi, R., Cabalar, P.: Solving a spatial puzzle using answer set programming integrated with markov decision process. In: 2018 7th Brazilian Conference on Intelligent Systems (BRACIS), pp. 528–533 (2018)
32. dos Santos, T.F., Santos, P.E., Ferreira, L.A., Bianchi, R.A.C., Cabalar, P.: Heuristics, answer set programming and markov decision process for solving a set of spatial puzzles. CoRR **abs/1903.03411** (2019). URL `http://arxiv.org/abs/1903.03411`
33. Sridharan, M., Gelfond, M., Zhang, S., Wyatt, J.: Reba: A refinement-based architecture for knowledge representation and reasoning in robotics. Journal of Artificial Intelligence Research **65**, 87–180 (2019)
34. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
35. Topin, N., Haltmeyer, N., Squire, S., Winder, J., desJardins, M., MacGlashan, J.: Portable option discovery for automated learning transfer in object-oriented markov decision processes. In: IJCAI, pp. 3856–3864 (2015)
36. Wałęga, P.A., Schultz, C., Bhatt, M.: Non-monotonic spatial reasoning with answer set programming modulo theories. Theory and Practice of Logic Programming **17**(2), 205–225 (2017)
37. Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**(3-4), 279–292 (1992)
38. Xiaomei, H., Jun, X., Jianfei, C.: Robot path planning based on an improved q-learning method. In: International Computer Science and Applications Conference (ICSAC 2019), pp. 99–102 (2019)
39. Yang, F., Lyu, D., Liu, B., Gustafson, S.: Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI'18, p. 4860–4866. AAAI Press (2018)
40. Zamani, M.A., Magg, S., Weber, C., Wermter, S., Fu, D.: Deep reinforcement learning using compositional representations for performing instructions. Paladyn, Journal of Behavioral Robotics **9**(1), 358 – 373 (2018)
41. Zhang, F., Duan, S., Wang, L.: Route searching based on neural networks and heuristic reinforcement learning. Cognitive neurodynamics **11**(3), 245–258 (2017)
42. Zhang, G., Li, Y., Xu, X., Dai, H.: Efficient training techniques for multi-agent reinforcement learning in combat tasks. IEEE Access **7**, 109301–109310 (2019)
43. Zhang, S., Sridharan, M., Wyatt, J.L.: Mixed logical inference and probabilistic planning for robots in unreliable worlds. IEEE Transactions on Robotics **31**(3), 699–713 (2015)