Solving a spatial puzzle using Answer Set Programming integrated with Markov Decision Process

Thiago Freitas dos Santos*, Paulo E. Santos*, Leonardo A. Ferreira†, Reinaldo A. C. Bianchi* and Pedro Cabalar‡
* Artificial Intelligence in Automation and Robotics (IAAAR)
Centro Universitário da FEI, São Bernardo do Campo - SP, Brazil
Email: thiagosantos38@gmail.com, psantos@fei.edu.br, rbianchi@fei.edu.br
† Accesstage Tecnologia S.A., São Bernardo do Campo - SP, Brazil
Email: leonardo.ferreira@accesstage.com.br
‡ Computing Department
University of Corunna, Corunna, Spain
Email: cabalar@udc.es

***Abstract*—Spatial puzzles are interesting domains to investigate problem solving, since the reasoning processes involved in reasoning about spatial knowledge is one of the essential items for an agent to interact in the human environment. With this in mind, the goal of this work is to investigate the knowledge representation and reasoning process related to the solution of a spatial puzzle, the Fisherman's Folly, composed of flexible string, rigid objects and holes. To achieve this goal, the present paper uses heuristics (obtained after solving a relaxed version of the puzzle) to accelerate the learning process, while applying a method that combines Answer Set programming (ASP) with Reinforcement learning (RL), the oASP(MDP) algorithm, to find a solution to the puzzle. ASP is the logic language chosen to build the set of states and actions of a Markov Decision Process (MDP) representing the domain, where RL is used to learn the optimal policy of the problem.**

***Index Terms*—reinforcement learning, spatial puzzle, answer set programming, heuristic, oASP(MDP)**

I. Introduction

Since the reasoning processes involved in reasoning about spatial knowledge is one of the essential items for an agent to interact in the human environment, it becomes interesting to investigate domains that incorporate these concepts, like spatial puzzles. Although reasoning about (ecological) space is straightforward for humans, this is not the case for automated methods, where the representation of spatial entities are elusive to be specified precisely. The present paper investigates spatial puzzles composed of rigid objects, flexible strings and holes, such as the Fisherman's Folly puzzle. In such puzzles, the complexity introduced by the manipulation of a string, as well as applying actions on holed objects, justifies its interest as a knowledge representation problem, and also as a challenging domain for automated problem solvers [1].

In order to equip artificial agents with the reasoning and learning capabilities necessary to interact with this domain, the present work uses the Online ASP for MDP (oASP(MDP)) algorithm [2], which models the domain as a Markov Decision Process (MDP) and has Answer Set Programming (ASP) as the function approximator. ASP is a language based in logic programming and non-monotonic reasoning, which provides a way to represent a domain with dynamic characteristics and to solve problems with common sense reasoning [3], [4]. MDP is a formalism used to describe a decision making problem, where Reinforcement Learning (RL) is a suitable tool to find optimal policies [5]. With the combination of these techniques, the oASP(MDP) algorithm is capable of updating previous learned policies with a RL method, while ASP builds the set of states, actions and transitions of the MDP. The contribution of the present paper, with respect to the previous applications of oASP(MDP), is the use of heuristic to accelerate the Reinforcement Learning procedure.

Finally, experiments (section IV) applying heuristics to the Fisherman's Folly puzzle demonstrates the viability of this approach, and its advantage compared to different reinforcement learning algorithms.

II. Background

A. Domain

The focus of this research is to find solutions to spatial puzzles composed of strings, posts, rings and regular objects. In this paper we present a solution to the Fisherman's Folly puzzle (Figure 1) whose goal is to free a ring from an entanglement of objects.

Toward a better understanding of the puzzle investigated in this paper, a diagrammatic representation is given in Figure 2, showing the elements that constitute the domain: one holed Post (fixed to a Base), one String, one Ring, two Disks: Disk1 and Disk2 (each one fixed to a different tip of the String) and two Spheres: Sphere1 and Sphere2 (each one is on one side of the Post and are crossed by the String). It is important to point out the possible actions that can be executed: the String can

978-1-5386-8023-0/18/$31.00 ©2018 IEEE
DOI 10.1109/BRACIS.2018.00097

Fig. 1. The Fisherman's Folly puzzle [6].

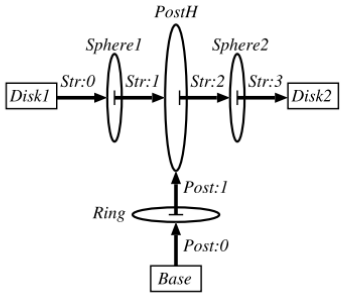


Fig. 2. A diagrammatic representation of the Fisherman's Folly puzzle [6].

pass through the Post hole with the manipulation of its tips. The Post and the two Spheres can pass through the Ring, while the Ring can pass through the Post hole, but only if the Post is not already crossing the Ring. The Disks cannot pass through the Ring, whereas the Spheres cannot pass through the Post hole. Another essential component when dealing with holes in this domain is that each hole has two faces (sides), one where the element starts to pass through the holed object, and the other is the face where the crossing element leaves this object.

To achieve the goal of releasing the ring, it is necessary to execute a sequence of actions, with the restriction that these actions cannot destroy any object of the puzzle. To complete this task, an initial state was defined (Figure 1) with the following description: The Post is crossing the Ring, while the String is fixed to Disk1, passing through three elements: Sphere1, Post hole and Sphere2, and then fixed to Disk2.

B. Reinforcement Learning (RL)

Some problems are hard to solve using only pre-defined rules, such as classification, regression or decision making problems in unknown environments. This justifies the use of Reinforcement Learning (RL), that is defined in [5] as a computational method for learning through the agent's interactions in a domain to achieve a goal. This is accomplished via the maximization (or minimization) of a numerical reward signal. A traditional RL algorithm has two different entities, one is the agent, responsible for learning the actions that lead to the solution of a problem, and the other is the environment, the place in which the agent executes actions. It is also important to define a formulation to describe this kind of problem, hence the use of Markov Decision Process (MDP), defined in [5] as a tuple <S, A, T, R>, where: [S] is the collection of possible states in the domain; [A] is the collection of actions that can be executed by the agent; [T] is the transition function (Equation (1)) responsible for providing the probability that the agent, present in a state s ∈ S and executing an action a ∈ A, will get to the future state s′ ∈ S; and [R] is the reward function (Equation (2)) responsible for providing the reward, when the agent is in a state s ∈ S and execute one action a ∈ A that leads to the future state s′ ∈ S.

$$\text{Transition} = S \times A \times S \mapsto [0, 1] \quad (1)$$

$$\text{Reward} = S \times A \times S \mapsto Re \quad (2)$$

After the definition of an MDP, it is important to choose an algorithm capable of finding an optimal policy. This paper uses a model-free off-policy algorithm that does not need *a priori* information about the transition and reward functions, the Q-Learning algorithm [5]. This algorithm can learn while the agent interacts with the environment. Q-Learning has an action-value function Q(s,a), used to define the policy. To update this action-value function, the algorithm runs a number of interactions of the agent with the environment, where each interaction happens in the following way: The agent executes an action a ∈ A at instant t and in a state s ∈ S. After this, the environment responds a future state s′ ∈ S at the instant t_{+1} and the reward r, updating the action-value function Q(s,a) using the formula in Equation (3). The additional terms of the formula "α" and "γ" represent the learning rate and the discount factor, respectively.

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \cdot (r + \gamma \cdot \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)) \quad (3)$$

After the end of the interactions the action-value function Q(s,a) has been updated (using the reward values) several times, leading to the optimal policy.

It is also possible to use heuristics to accelerate the approximation of the action-value function Q(s,a), as presented in [7]. A heuristic is responsible for helping to guide the learning process, since it is used in the action selection phase of the RL algorithm. [7] points out that the only difference between the standard implementation of Q-Learning and the Heuristically Accelerated Q-Learning is in the action selection rule. The action selection rule is shown in Equation 4:

$$\pi(s) = \begin{cases} argmax_a[\hat{Q}(s,a) + \xi H(s,a)^{\beta}], & \text{if } q \leq p. \\ a_{random}, & \text{otherwise,} \end{cases} \quad (4)$$

where H(s,a) is the heuristic function that guides the choice of action, ξ and β are parameters that control the influence of the heuristic function, q is a random value, between 0 and 1, p is a value, between 0 and 1, responsible for defining the exploitation/exploration trade-off in the Q-Learning algorithm. a_{random} is an action that was chosen randomly.

To allow the interaction between the agent and the environment, the PROLOG simulation presented in [6] is used in this paper as an oracle with which the RL agent interacts, since it allows the type of queries necessary to determine the future

state of the puzzle when an action is executed. For example, the RL agent may ask the PROLOG program what would be the future state of the puzzle when an action is executed, the program then returns two possible answers to the agent, either the successor state of the puzzle or the information that it is impossible to execute that action.

C. Answer Set Programming (ASP)

Answer Set Programming (ASP) is a declarative language based on logic programming and non-monotonic reasoning that is an efficient tool for solving NP-Complete problems [3]. ASP is also very useful when a problem involves commonsense reasoning, as the domain investigated by this work.

An ASP program is a set of Horn clauses of the form:

```
A :- L1, L2,..., Ln          (5)
```

in which A is an atom (or the head of the clause), and the conjunction of literals, `L1, L2, ..., Ln` is the body [8].

An interesting possibility when using ASP is the generation of different outcomes when the same input is given. This can be achieved thanks to the use of choice rules, described as:

```
1{s1, s2, s3 }1 :-a, s.          (6)
```

where, given the current state s ∈ S and an action a ∈ A, this rule has as possible outcomes the states s1, s2 and s3 ∈ S. For each state s ∈ S there is an ASP program describing the consequence of executing each allowed action a ∈ A. Domain constraints can also be represented with ASP programs, in this context, these constraints are related to the allowed and forbidden states, action and state/action pairs. In conclusion, finding all answer sets for every state that the agent is allowed to visit is the same as finding a set of states of an MDP, i.e., to find every allowed transition for each state-action pair [2].

D. Online ASP for MDP (oASP(MDP))

The Online ASP for MDP (oASP(MDP)) algorithm, proposed in [2], aims to combine the advantages of ASP and RL, in order to find the set of states S with ASP and to approximate an action-value function Q(s,a) of an MDP, through the application of the Q-Learning method. With this combination it is possible to describe choice rules for the description of the transition function t(s, a, s') in the logical form, as follows:

```
1{s'}1 :- a.          (7)
```

According to [2], one of the main advantages of this logical description is the possibility of applying it to every action and state of the domain, modeling the possible transitions to each state as a logical program. Then, an ASP engine is used to get a set of observable states and actions, this is done by finding every answer set for every state s that is being visited in a given instant. After this step, and through numerous interactions between the agent and the environment, the oASP(MDP) algorithm is capable of approximating the action-value function Q(s,a).

A pseudo code representation of the oASP(MDP) algorithm in shown in Algorithm 1. First, the algorithm receives three distinct parameters, the first is the set of actions the agent can execute; the second is a RL method M that can be used to approximate the action-value function Q(s,a); and, finally, it receives the number of times n that the agent is going to interact with the environment. After receiving these initial parameters, the algorithm initializes the set of states S as empty, since the construction of this set of states is done incrementally as the interactions occur. The main contribution of this algorithm is that, even though it builds the set of states in an incremental way, the RL method M is still able to approximate the action-value function Q(s,a) [2].

```
1  Algorithm: oASP(MDP)
   Input: The set of actions 𝒜, an action-value function
          approximation method M and a number of
          episodes n.
   Output: The approximated Q(s,a) function.
2  Initialize the set of observed states 𝒮_o = ∅
3  while number of episodes performed is less than n do
4      repeat
5          Observe the current state s
6          if s ∉ 𝒮_o then
7              Add s to the set of states 𝒮_o.
8              Choose and execute a random action a ∈ 𝒜.
9              Observe the future state s'.
10             Update state s logic program with observed
               transition adding a choice rule.
11             Update Q(s,a)'s description by finding every
               answer set for each state s added to 𝒮_o in
               this episode.
12         else
13             Choose an action a ∈ 𝒜 as defined by M.
14             Execute the chosen action a.
15             Observe the future state s'.
16         Update Q(s,a)'s value as defined by M.
17         Update the current state s ← s'.
18     until the end of the episode
```

Algorithm 1: The oASP(MDP) Algorithm created by [2].

The next step of the algorithm is a loop where, in each repetition, the algorithm observes the current state s of the agent and if $s \notin S$, then s is added to S, one random action is executed and a future state s' is returned. After this, it is possible to create the choice rules (such as Formula (7)) and to update the action-value function Q(s,a) according to the received reward r. On the other hand, if s ∈ S, then another decision is made since, in this case, the action to be executed is chosen by the RL method M [2].

Finally the action-value function Q(s,a) is updated for each step of the episode. This update is done by M. The current state is also updated in this step, since now the current state is s' [2].

The next section describes the main contributions of the present paper.

III. Solving a spatial puzzle with oASP(MDP)

In order to solve spatial puzzles of increasing complexity, this work adapted the oASP(MDP) algorithm to use heuristics to accelerate the learning process.

To obtain an admissible heuristic for the Fisherman's Folly, Q-Learning was applied to a simplified version of this puzzle, where multiple passes of the string through holes were forbidden, whereas the arrangement of objects was kept the same as in the original puzzle. Thus, the Q-table obtained as a solution to this simpler puzzle was used as heuristic to guide the choice of an action to be executed, accelerating the learning process, in the original Fisherman's Folly puzzle.

We now describe each part of the domain formulation and its use in oASP(MDP).

First, actions in the domain were defined such that the agent is capable of passing objects through holes, this is represented as a tuple `<HE, CE, HF>`, where:

HE is a Host Element, i.e., an element of the puzzle that could host a hole[1]. $HE \in \{Sphere1, Sphere2, Post, Disk1, Disk2, Ring, String\}$

CE is a Crossing Element, an element of the puzzle that is going to pass through a hole in a Host Element. The set of CEs includes all the objects in the puzzle, even though the String only crosses an object with its two tips: Str1, Str2 (fixed to Disk1 and Disk2, respectively); $CE \in \{Sphere1, Sphere2, Post, Disk1, Disk2, Ring, String\}$

HF is the Hole Face, the side of the hole towards which the CE is going be passed through the hole in HE. There are two possible sides (or faces): positive (+) or negative (-).

Thus, the tuple `<HE, CE, HF>` represents that "CE is going to pass through HE towards HF". Choosing an action is to choose the elements that the agent is going to manipulate in the interaction. There are 7 elements the agent is capable of manipulating as a CE, 7 elements as a HE and 2 HF, which leads to the total of 98 possible actions.

Note that, initially, the agent does not have the knowledge about which objects have holes, and which do not have (thus, the set HE contains all domain objects); nor the agent knows beforehand which elements can or cannot pass through existing holes (due to size constraints). These are characteristics of the domain that are going to be learned by the RL procedure, by assigning negative rewards to unfruitful actions, such as trying to pass an object through the Disks (which host no holes).

The domain states are represented as a list structure $chain$ (similar to that introduced in [1]) to capture the sequence of holes that any object is currently crossing. In this work, we only assume this list representation for the string and for the post. For instance, each crossing of a string s through a hole h is represented by the exit hole face: if s crosses h from $-h$ to $+h$, we represent the crossing as $+h$ in the chain representation.

For example, consider the diagrammatic representation of the Fisherman's Folly puzzle in Figure 2, the lists of crossings for the string (str) and the post are the following:

1) $chain(str) = [+Sphere1, +Post, +Sphere2]$
2) $chain(post) = [+Ring]$

After defining the representation for the actions and states, the oASP(MDP) can be initialized. It receives the set of actions A, the set of goal states S and the number of episodes n.

In the first step of the algorithm, the agent needs to verify if the initial state of the puzzle ($s0$) is already in set S. At the start of the process, the initial state is not in the set, leading the agent to execute a random action, for example, pass the disk through the post toward the positive face: `<Post Hole, Disk, Positive>`. After this execution, the agent goes to state ($s1$), then $s0$ can be added to S and the agent receives a reward $r0$. Now the agent has knowledge about a possible transition, which can be translated to a choice rule in ASP as `1{s1}1 :- a(ExecutedAction)`, where "ExecutedAction" is the action `<Post Hole, Disk, Positive>` executed by the agent. Each state $s \in S$ has an ASP file with all these transitions rules. Succeeding the creation of this file, ASP can find the answer sets and the agent can initialize the action-value function Q(s,a), updating the value with the received reward $r0$.

Now the agent is in $s1$, and since $s1 \notin S$, all the steps described above are repeated. But, another situation may happen, let's assume that the action chosen randomly now is to undo the crossing of the disk through the post hole `<Post Hole, Disk, Negative>`, this action leads again to the initial state $s0$. Because the agent is in $s0$, it is not going to execute a random action, since it has now an initial value for Q(s,a). The agent is going to execute an action determined by the Q-Learning algorithm, leading to the future state s'.

The reward used by the agent in order to update the action-value function Q(s,a) is returned by the environment. Finally, after a determined number of interactions, the oASP(MDP) algorithm is capable of returning the Q-table. This table represents the values for each action in the states visited by the agent, one property of the Q-table is that if the agent executes the actions with the highest Q-values, the agent is executing the optimal policy [5].

IV. Experiments and Results

The Fisherman's Folly puzzle was tested with four distinct reinforcement learning algorithms: the traditional version of Q-Learning, oASP(MDP) [2], Q-Learning accelerated by heuristic [7] and oASP(MDP) with the RL method accelerated by heuristic (first proposed in this paper).

The RL parameters used throughout this work were: discount factor of 0.9; the trade-off between exploitation and exploration was fixed to 0.1, throughout the whole experiment, the learning rate was 0.2 and the control value for the heuristic was 0.1. Related to the reward received by the agent, the following values were defined: -100 to forbidden actions

[1]For brevity we identify the hole in an object by the name of the object itself.

(actions that lead to the same state or that are physically impossible), 100 to reaching the goal and -5 to each step executed.

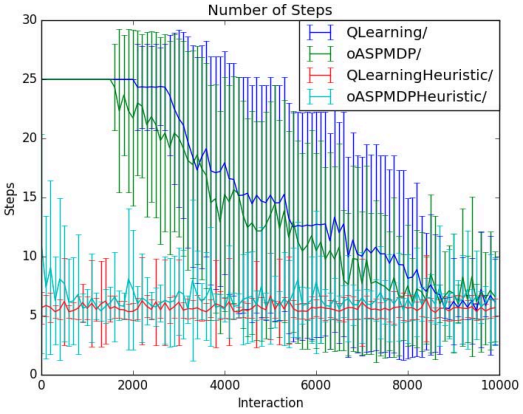
To evaluate these approaches, we used four different measures. The first is the number of steps to complete the puzzle, the optimal value is five. Second, the return values, with the maximum value of 80 to the optimal case. The last two were: the number of states visited and the number of state/action pair in the Q-Table. For each of the 30 trials, 10 thousand episodes were executed. An episode ends when the agent reaches the goal or the number of steps is 25.

Results show that the heuristically accelerated versions of RL are better suited than non-accelerated versions, since the former guided the learning of the algorithms from the early beginning, reaching the optimal policy (Figure 3) faster and also presenting higher return values (Figure 4). This happens because the heuristic used by this paper is the Q-table of the solution of a relaxed version of the puzzle.

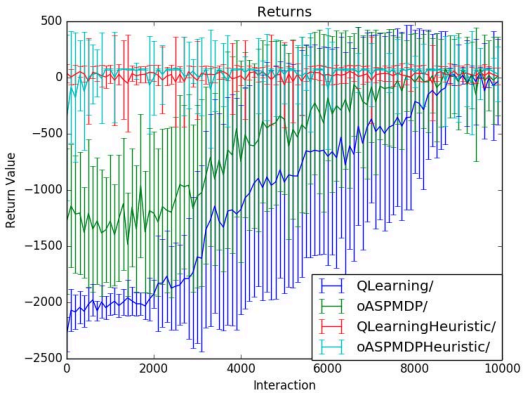
From the beginning of the interactions, the traditional version of Q-Learning already has the full Q-table, with all actions that can be chosen by the agent. Different from the oASP(MDP) algorithm, that builds this table through interactions. This characteristic leads to an interesting comparison between the number of states visited (Figure 5) and the number of state/action pairs (Figure 6). The oASP(MDP) algorithm visits more valid states than the traditional Q-Learning, although it has a smaller number of state/action pairs because of the removal of forbidden actions. The oASP(MDP) algorithm does not even try to perform these actions, not adding these to the Q-table, different from the Q-Learning that has all the actions on the Q-table from the beginning. Besides that, Q-Learning also does not remove forbidden actions from the list of possible actions to be performed, receiving only negative rewards as an indication of how bad an action is, so Q-Learning tries to execute these forbidden actions in future steps, which leads to the same state and consequently (in overall) the number of states that are explored end up being smaller than the oASP(MDP).

But one drawback of oASP(MDP) is that it requires more time to run than the traditional Q-Learning. Besides, ASP capabilities are not fully explored due to the simplicity of the domain. So, future work will focus on expanding the use of ASP.

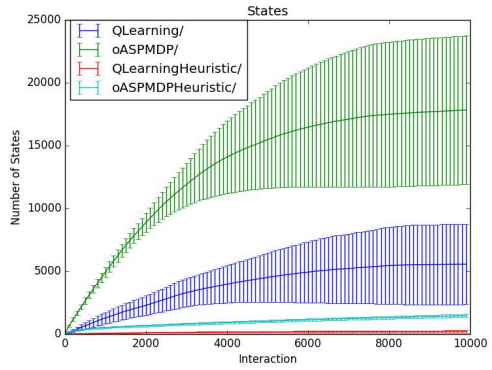
The algorithms were implemented in Python 3.5, using ZeroMQ to provide message exchange between the environment and the agent. SWI-Prolog was the PROLOG environment and Clingo was the ASP engine. The source code for the tests can be found at: https://goo.gl/7wzZ56



(3) Number of steps.



(4) Return values.

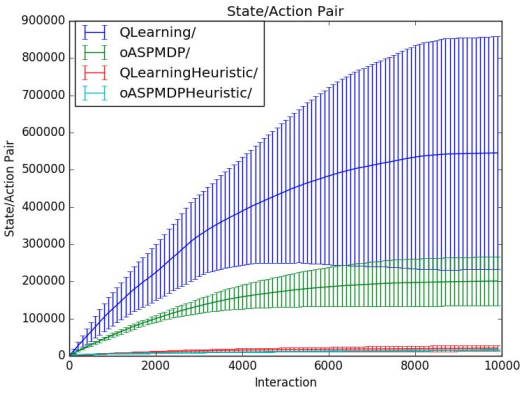


(5) Number of states visited.

V. Literature Review

The work in [9] explores the combination of ASP and RL, presenting the DARLING framework. This framework uses ASP and CLINGO in order to represent models and to allow planning and reasoning, while uses RL to make the agent adaptive to the environment. DARLING was tested with a service robot in an office-like environment, allowing the authors to show that when DARLING is used, the robot can learn tasks faster and improve its performance over time. One aspect that differentiate DARLING from oASP(MDP) is the reliance of a planning phase in the former (not needed in the latter). Future work shall explore this distinction.

The work reported in [10] presents an approach that combines Deep Q-Network with Symbolic Representation, where the latter is used as input and output of the deep learning algorithm. The representation in [10] describes spatial relations between objects and the number of possible combinations of



(6) Number of state/action pairs.

these predicates makes the Q-table an interactive model.

The present paper adapts oASP(MDP) [2] to deal with heuristics. The work reported in [7] introduces an interesting approach, the Heuristic Accelerate Reinforcement Learning (HARL) algorithm, where heuristics were used to accelerate the RL learning process. After this initial proposal, different approaches in different domains were explored. In [11], the combination of heuristic and RL is used on the domain of dynamic secondary spectrum sharing in cellular systems. This is an interesting study because it compares HARL to RL and Heuristic systems alone, presenting how HARL can outperform these other two approaches. The suitability of HARL can also be seen in two robot domains [12], where case base was used as heuristic in a transfer learning setting. This approach works as follows: first a RL algorithm is applied to a task, then when the learning converges, a database of cases is stored. These cases are then transferred to be used in another task, where the cases are retrieved and adapted. The present paper uses a similar idea, first, it runs the Q-Learning algorithm to a relaxed version of the problem, then it uses the Q-Table as heuristic to the complete Fisherman's Folly puzzle.

There are also solutions of spatial puzzles from a more formal knowledge representation standpoint.

A formalization of the Fisherman's Folly puzzle was described in [6], where a version of Equilibrium Logic and Situational Calculus are taken into consideration, using the former to represent the semantics, and the latter to provide a tool to formalize the problem. Besides that, a PROLOG planner, used to solve the Fisherman's Folly puzzle, is presented. One drawback is that this planner is not suitable to solve more complex spatial puzzles, as the representation (and, therefore, the complexity of the state space) is prohibitively high. This is the main motivation for the development of the work reported in this paper, the combination of ASP with RL allows for a partial representation of the domain to trim the search space of an agent that learns the solution of complex problems by interacting with the environment.

VI. Conclusion

This paper presented the application of different Reinforcement Learning techniques to the Fisherman's Folly puzzle domain. The oASP(MDP) is one of these techniques, where a combination of MDP and Answer Set Programming is achieved. The RL method used by this algorithm is the off-policy model-free Q-Learning.

Experiments were performed in order to compare the different RL approaches: traditional Q-Learning, oASP(MDP), Q-Learning accelerated by heuristic and oASP(MDP) with the Q-Learning method accelerated by heuristic. The results showed how heuristics can accelerate the learning process, since it always outperforms its non-heuristics counterparts. Besides, the results also presented how an expanded exploration of the domain is achieved with oASP(MDP).

Acknowledgment

Thiago Freitas is sponsored by FAPESP-IBM Proc. 17/07833-9. Paulo E. Santos and Leonardo Ferreira acknowledge financial support from FAPESP-IBM Proc. 2016/18792-9. Reinaldo Bianchi acknowledges financial support from FAPESP Proc. 2016/21047-3.

References

[1] P. E. Santos and P. Cabalar, "Framing holes within a loop hierarchy," *Spatial Cognition & Computation*, vol. 16, no. 1, pp. 54–95, 2016.

[2] L. A. Ferreira, R. A. d. C. Bianchi, P. E. Santos, and R. L. De Mantaras, "A method for the online construction of the set of states of a markov decision process using answer set programming," 2018.

[3] T. Eiter, G. Ianni, and T. Krennwallner, "Answer set programming: A primer," in *Reasoning Web. Semantic Technologies for Information Systems*. Springer, 2009, pp. 40–110.

[4] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming." in *ICLP/SLP*, vol. 88, 1988, pp. 1070–1080.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning an introduction – Second edition, in progress (Complete Draft)*. MIT Press, 2018.

[6] P. Cabalar and P. E. Santos, "Formalising the fisherman's folly puzzle," *Artificial Intelligence*, vol. 175, no. 1, pp. 346–377, 2011.

[7] R. A. C. Bianchi, C. H. C. Ribeiro, and A. H. R. Costa, "Heuristically accelerated q–learning: A new approach to speed up reinforcement learning," in *Advances in Artificial Intelligence – SBIA 2004*, A. L. C. Bazzan and S. Labidi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 245–254.

[8] V. Lifschitz, "What is answer set programming?." in *AAAI*, vol. 8, no. 2008, 2008, pp. 1594–1597.

[9] M. Leonetti, L. Iocchi, and P. Stone, "A synthesis of automated planning and reinforcement learning for efficient, robust decision-making," *Artificial Intelligence*, vol. 241, pp. 103 – 130, 2016.

[10] M. A. Zamani, S. Magg, C. Weber, and S. Wermter, "Deep reinforcement learning using symbolic representation for performing spoken language instructions," in *2nd Workshop on Behavior Adaptation, Interaction and Learning for Assistive Robotics (BAILAR) on Robot and Human Interactive Communication (RO-MAN), 26th IEEE International Symposium on*, 2017.

[11] N. Morozs, T. Clarke, and D. Grace, "Heuristically accelerated reinforcement learning for dynamic secondary spectrum sharing," *IEEE Access*, vol. 3, pp. 2771–2783, 2015.

[12] R. A. C. Bianchi, P. E. Santos, I. J. da Silva, L. A. Celiberto, and R. Lopez de Mantaras, "Heuristically accelerated reinforcement learning by means of case-based reasoning and transfer learning," *Journal of Intelligent & Robotic Systems*, Oct 2017.