Imperial College London

Department of Computing

# Symbolic Reinforcement Learning using Inductive Logic Programming

*Author:*
Kiyohito Kunii

*Supervisor:*
Prof. Alessandra Russo
Mark Law
Ruben L Vereecken

Submitted in partial fulfillment of the requirements for the MSc degree in MSc in Computing Science of Imperial College London

June 2018

## Abstract

Your abstract.

# Acknowledgments

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

There have been successful applications of deep reinforcement learning (DRL) in a number of domains, such as video games [**?** ], the game of Go [**?** ] and robotics [**?** ]. However, there are still a number of issues to overcome with this method. First, it requires large dataset for training the model, and the learning is very slow and requires significant amount of computation. Second, it is considered to be a black-box, meaning that the decision making process is unknown to the human user and therefore lacks explanation of the decision making. Third, there is no thought process to the decision making, such as understanding relational representations or planning. To tackle these problems, researchers have explored several different approaches. One of the methods is to incorporate symbolic representations into the system [**?** ]. This approach is promising and shows a potential.

In this paper, we extend this symbolic representation approach and explore the potential of symbolic machine learning to solve the above issues. There are several advantages of symbolic machine learning. First of all, the decision making mechanism is understandable by humans rather than being black-box. Second, it resembles how humans reason. Similar to reinforcement learning, there are some aspects of trial-and-error in human learning, but humans exploit reasonings to efficiently learn about their surrounding or situations. They also effectively use previous experience (e.g background knowledge) when encountering similar situations. Finally, the recent advance of Inductive Logic Programming (ILP) research has enabled us to apply ILP in more complex situations and there are a number of new algorithms based on Answer Set Programmings (ASPs) that work well in non-monotonic scenarios.

Particularly since [**?** ], there have been several researches that further explored the incorporation of symbolic reasoning into RL, but the combining of ILP and RL has not been explored. Because of the recent advancement of ILP and RL, it is natual to consider that a combination of both approaches would be the next field to explore.

In this paper, our objective is to explore the incorporation of ILP into RL using Inductive Learning of Answer Set Programs (ILASP), which is a state-of-art ILP method that can be applied to incomplete and more complex environments.

TODO Update this

This background report will be part of the final report and is organised as follows: In Chapter **??**, the background of inductive logic programming and reinforcement learning necessary for this paper are described. Chapter **??** discusses previous research on relevant approach. Chapter **??** shows the tentative architecture of our new approach, using ILASP to generate a model of the environment. We also discuss some of the issues we currently face with the architecture and plan the implementation. Finally the ethics checklist is provided in Chapter **??**.

# Chapter 2

# Background

This chapter introduces necessary background of Inductive Logic Programming (Section **??**) and Reinforcement Learning (Section **??**), which provide the foundations of our research.

## 2.1 Inductive Logic Programming (ILP)

*Inductive Logic Programming (ILP)* is a subfield of machine learning research area aimed at the intersection between machine learning and logic programming [**?** ]. The purpose of ILP is to inductively derive a hypothesis H that is a solution of a learning task, which coveres all positive examples and none of negative examples, given a hypothesis language for search space and cover relation [**?** ]. ILP is based on learning from entailment, as shown in Equation **??**.

$$B \wedge H \models E \tag{2.1}$$

where E contains all of the positive examples ($E^+$) and none of the negative examples ($E^-$). One of the advantage of ILP over statistical machine learning is that the hypothesis that an agent learnt can be easily understood by a human, as it is expressed in first-order logic, making the learning process more transparent rather than blackbox. One of the limitations of ILP is learning efficiency and scalability. There are usually thousands or more examples in many real-world examples. Scaling ILP task to cope with large examples is a challenging task [**?** ].
In this section, we briefly introduce foundation of Answer Set Programming (ASP) and inductive learning frameworks.

### 2.1.1 Stable Model Semantics

Having defined the syntax of clausal logic, we now introduce its semantics under the context of Stable Model. The semantics of the logic is based on the notion of interpretation, which is defined under a *domain*. A domain contains all the objects that exist. In logic, it is convention to use a special interpretations called *Herbrand interpretations* rather than general interpretations.

**Definition 2.1.** *Herbrand Domain* (a.k.a *Herbrand Universe*) of clause sets *Th* is the set of all ground terms that are constants and function symbols appeared in *Th*.

**Definition 2.2.** *Herbrand Base* of *Th* is the set of all ground predicates that are formed by predicate symbols in *Th* and terms in the Herbrand Domain.

**Definition 2.3.** *Herbrand Interpretation* of a set of definite clauses *Th* is a subset of the Herbrand base of *Th*, which is a set of ground atoms that are true in terms of interpretation.

**Definition 2.4.** *Herbrand Model* is a Herbrand interpretation if and only if a set *Th* of clauses is satisfiable. In other words, the set of clauses *Th* is unsatisfiable if no Herbrand model was found.

**Definition 2.5.** *Least Herbrand Model* (denoted as *M(P)*) is an unique minimal Herbrand model for definite logic programs. The Herbrand Model is a minimum Herbrand model if and only if none of its subsets is an Herbrand model.

For normal logic programs, there may not be any least Herbrand Model.

**example 2.1.1.** (Herbrand Interpretation, Herbrand Model and M(P))

$$\mathrm{P} = \begin{cases} p(X) \leftarrow q(X) \\ q(a). \end{cases} \qquad \mathrm{HD} = \{\ \mathrm{a}\ \}\ ,\ \mathrm{HB} = \{\ \mathrm{q(a)},\ \mathrm{p(a)}\ \}$$

where HD is Herbrand Domain and HB is Herbrand Base. Given above, there are four Herbrand Interpretations = ⟨ {q(a)}, {p(a)}, {q(a), p(a)}, {} ⟩, and one Herbrand Model (as well as M(P)) = {q(a), p(a)}

*Definite Logic Program* is a set of definite rules, and a *definite rule* is of the form $h \leftarrow a_1, ..., a_n$. $h$ and $a_1, ..., a_n$ are all atoms. $h$ is the *head* of the rule and $a_1, ..., a_n$ are the *body* of the rule. *Normal Logic Program* is a set of normal rules, and a normal rule is of the form $h \leftarrow \mathrm{a}_1, ..., a_n$, *not* $b_1$, ..., *not* $b_n$ where $h$ is the head of the rule, and $a_1, ..., a_n, b_1, ..., b_n$ are the body of the rule (both the head and body are all atoms).
To solve a normal logic program *Th*, the program P needs to be grounded. The *grounding* of *Th* is the set of all clauses that are c ∈ *Th* and variables are replaced by terms in the Herbrand Domain.

**Definition 2.6.** The algorithm of grounding starts with an empty program Q = { } and the relevant grounding is constructed by adding to each rule R to Q such that

- R is a ground instance of a rule in P.
- Their positive body literals already occurs in the in the of rules in Q.

The algorithm terminates when no more rules can be added to Q.

**example 2.1.2.** Grounding

$$\mathrm{P} = \begin{cases} q(X) \leftarrow p(X). \\ p(a). \end{cases}$$

ground(P) in this example is {p(a), q(a)}.

Not only the entire program needs to be grounded in order for an ASP solver to work, but also each rule must be *safe*. A rule *R* is safe if every variable that occurs in the head of the rule occurs at least once in body$^+$(R) . Since there is no unique least Herbrand Model for a normal logic program, Stable Model of a normal logic program was defined in [**?** ]. In order to obtain the Stable Model of a program P, P needs to be converted using *Reduct* with respect to an interpretation X.

**Definition 2.7.** The *reduct* of P with respect to X can be constructed such that

- If the body of any rule in P contains an atom which is not in X, those rules need to be removed.
- All default negation atoms in the remaining rules in P need to be removed.

**example 2.1.3.** Reduct

$$P = \begin{cases} p(X) \leftarrow not\ q(X). \\ q(X) \leftarrow not\ p(X). \end{cases}, X = \{p(a), q(b)\}$$

Where X is a set of atoms. ground(P) is
p(a) ← not q(a).
p(b) ← not q(b).
q(a) ← not p(a).
q(b) ← not p(b).

The first step removes p(b) ← not q(b). and q(a) ← not p(a).
p(a) ← not q(a).
q(b) ← not p(b).

The second step removes negation atoms from the body.
Thus reduct $P^x$ is $(\text{ground}(P))^x = \{p(a), q(b).\}$

A Stable Model of P is an interpretaiton X if and only if X is the unique least Herbrand Model of ground$(P)^x$ in the logic program.

### 2.1.2 Anwer Set Programming (ASP) Syntax

**Definition 2.8.** Answer set of normal logic program P is a Stable Model, and Answer Set Programming (ASP) is a normal logic program with extensions: constraints, choice rules and optimisation statements. ASP program consists of a set of rules, where each rule consists of an atom and literals.

A *constraint* of the program P is of the form $\leftarrow a_1, ..., a_n, \textit{not } b_1, ..., \textit{not } b_n$, where the rule has an empty head. The constraint filters any irrelevant answer sets. When computing ground$(P)_x$, the empty head becomes ⊥, which cannot be in the answer sets. There are two types of constraints: *hard constraints* and *soft constraints*. Hard constraints are strictly satisfied, whereas soft constraints may not be satisfied but the sum of the violations should be minimised when solving ASP.

A *choice rule* can express possible outcomes given an action choice, which is of the form $l\{h_1,...,h_m\}u \leftarrow a_1, ..., a_n, \text{not } b_1, ..., \text{not } b_n$ where l and u are integers and $h_i$ for $1 \leq i \leq m$ are atoms. The head is called *aggregates*.
*Optimisation statement* is useful to sort the answer sets in terms of preference, which is of the form *#minimize[$a_1$=$w_1$,...$a_n$=$w_n$]* or *#maximize[$a_1$=$w_1$,...$a_n$=$w_n$]* where $w_1$,..., $w_n$ is integer weights and $a_1$,...,$a_n$ is ground atoms. ASP solvers compute the scores of the weighted sum of the sets of ground atoms based on the true answer sets, and find optimal answer sets which either maximise or minimise the score.
*Clingo* is one of the modern ASP solvers that executes the ASP program and returns answer sets of the program ([**?** ]), and we will use *Clingo* for the implementation of this research.

### 2.1.3 ILP under Answer Set Semantics

There are several ILP non-monotonic learning frameworks under the answer set semantics . We first introduce two of them: *Cautious Induction* and *Brave Induction* ([**?** ]), which are foundations of *Learning from Answer Sets* discussed in Section **??**, a state-of-art ILP framework that we will use for our research. (for other non-monotonic ILP frameworks, see [**?** ], [**?** ], [**?** ] and [**?** ]).

#### Cautious Induction

Cautious Induction task [1] is of the form $\langle$ B, $E^+$, $E^-$ $\rangle$, where B is the background knowledge, $E^+$ is a set of positive examples and $E^-$ is a set of negative examples.
H $\in$ $ILP_{cautious}$ $\langle$ B, $E^+$, $E^-$ $\rangle$ if and only if there is at least one answer set A of B $\cup$ H (B $\cup$ H is satisfiable) such that for every answer set A of B $\cup$ H:

1. $\forall$ e $\in$ $E^+$ : e $\in$ A
2. $\forall$ e $\in$ $E^-$ : e $\notin$ A

**example 2.1.4.** Cautious Induction

$$B = \begin{cases} exercises \leftarrow not\ eat\_out. \\ eat\_out \leftarrow exercises. \end{cases} \quad E^+ = \{\text{tennis}\},\ E^- = \{\text{eat\_out}\}$$

One possible H $\in$ $ILP_{cautious}$ is {tennis← exercises, ← not tennis }.

The limitation of Cautious Induction is that positive examples must be true for all answer sets and negative examples must not be included in any of the answer sets. These conditions may be too strict in some cases, and Cautious Induction is not able to accept a case where positive examples are true in some of the answer sets but not all answer sets of the program.

[1] This is more general definition of Cautious Induction than the one defined in [**?** ], as the concept of negative examples was not included in the original definition.

**example 2.1.5.** Limitation of Cautious Induction

$$B = \begin{cases} 1\{situation(P, awake), situation(P, sleep)\}1 \leftarrow person(P). \\ person(john). \end{cases}$$

Neither of *situation(john, awake)* nor *situation(john, sleep)* is false in all answer sets. In this example, it only returns person(john). Thus no examples could be given to learn the choice rule.

### Brave Induction

Brave Induction task is of the form $\langle$ B, $E^+$, $E^-$ $\rangle$ where, B is the background knowledge, $E^+$ is a set of positive examples and $E^-$ is a set of negative examples. H $\in$ $ILP_{brave}$ $\langle$ B, $E^+$, $E^-$ $\rangle$ if and only if there is at least one answer set A of B $\cup$ H such that:

1. $\forall$ e $\in$ $E^+$ : e $\in$ A

2. $\forall$ e $\in$ $E^-$ : e $\notin$ A

**example 2.1.6.** Brave Induction

$$B = \begin{cases} exercises \leftarrow not\ eat\_out. \\ tennis \leftarrow holiday \end{cases} \quad E^+ = \{\text{tennis}\}, E^- = \{\text{eat\_out}\}$$

One possible H $\in$ $ILP_{brave}$ is {tennis}, which returns {tennis, holidy, exercises} as answer sets.

The limitation of Brave Induction that it cannot learn constraints, since the above conditions for the examples only apply to at least one answer set A, whereas constrains rules out all answer sets that meet the conditions of the Brave Induction.

**example 2.1.7.** Limitation of Brave Induction (Example)

$$B = \begin{cases} 1\{situation(P, awake), situation(P, sleep)\}1 \leftarrow person(P). \\ person(C) \leftarrow super\_person(C). \\ super\_person(john). \end{cases}$$

In order to learn the constraint hypothesis H = { $\leftarrow$ not situation(P, awake), super_person(P)}, it is not possible to find an optimal solution.

### 2.1.4 Inductive Learning of Answer Set Programs (ILASP)

**Learning from Answer Sets (LAS)**

*Learning from Answer Sets (LAS)* was developed in [**?** ] to faciliate more complex learning tasks that neither Cautious Induction nor Brave Induction could learn. Examples used in LAS are *Partial Interpretations*, which are of the form $\langle$ $e^{inc}$, $e^{exc}\rangle$. (called *inclusions* and *exclusions* of *e* respectively). A Herbrand Interpretation extends a partial interpretation if it includes all of $e^{inc}$ and none of $e^{exc}$. LAS is of the form $\langle$ B, $S_M$, $E^+$, $E^-$ $\rangle$, where B is background knowledge, $S_M$ is hypothesis space, and $E^+$ and $E^-$ are examples of positive and negative partial interpretations. $S_M$ consists of a set of normal rules, choice rules and constraints. $S_M$ is specified by *language bias* of the learning task using *mode declaration*. Mode declaration specifies what can occur in a hypothesis by specifying the predicates, and consists of two parts: *modeh* and *modeb*. *modeh* and *modeb* are the predicates that can occur in the head of the rule and body of the rule respectively. Language bias is the specification of the language in the hypothesis in order to reduce the search space for the hypothesis.

**Definition 2.9.** Learning from Answer Sets (LAS)
Given a learning task T, the set of all possible inductive solutions of T is denoted as $ILP_{LAS}$(T), and a hypothesis H is an inductive solution of $ILP_{LAS}$(T) $\langle$ B, $S_M$, $E^+$, $E^-$ $\rangle$ such that:

1. H $\subseteq$ $S_M$
2. $\forall$ e $\in$ $E^+$ : $\exists$ A $\in$ Answer Sets(B $\cup$ H) such that A extends e
3. $\forall$ e $\in$ $E^-$ : $\nexists$ A $\in$ Answer Sets(B $\cup$ H) such that A extends e

**Inductive Learning of Answer Set Programs (ILASP)**

*Inductive Learning of Answer Set Programs (ILASP)* is an algorithm that is capable of solving LAS tasks, and is based on two fundamental concepts: *positive solutions* and *violating solutions*.
A hypothesis H is a positive solution if and only if

1. H $\subseteq$ $S_M$
2. $\forall$ $e^+$ $\in$ $\exists$ A $\in$ Answer Sets(B $\cup$ H) such that A extends $e^+$

A hypothesis H is a violating solution if and only if

1. H $\subseteq$ $S_M$
2. $\forall$ $e^+$ $\in$ $E^+$ $\exists$ A $\in$ Answer Sets(B $\cup$ H) such that A extends $e^+$
3. $\exists$ $e^-$ $\in$ $E^-$ $\exists$ A $\in$ Answer Sets(B $\cup$ H) such that A extends $e^-$

Given both definitions of positive and violating solutions, $ILP_{LAS}$ $\langle$ B, $S_M$, $E^+$, $E^-$ $\rangle$is positive solutions that are not violating solutions.

### A Context-dependent Learning from Answer Sets

*Context-dependent learning from ordered answer sets ($ILP_{LOAS}^{context}$)* is a further generalisation of $\mathrm{ILP_{LOAS}}$ with *context-dependent examples* [**?** ] Context-dependent examples are examples that each unique background knowledge (context) only applies to specific examples. This way the background knowledge is more structured rather than one fixed background knowledge that are applied to all examples. Formally, partial interpretation is of the form ⟨ e, C ⟩ (called *context-dependent partial interpretation (CDPI)*), where *e* is a partial interpretation and C is called *context*, or an ASP program without weak constraints. A *context-dependent ordering example (CDOE)* is of the form ⟨ ⟨$e_1$, $C_1$ ⟩, ⟨ $e_2$, $C_2$ ⟩⟩, which is a pair of CDPI. An APS program P *bravely respects o* if and only if

1. ∃ ⟨ $A_1$, $A_2$ ⟩ such that $A_1$ ∈ Answer Sets(P ∪ $C_1$), $A_2$, ∈ Answer Sets(P ∪ $C_2$), $A_1$ extends $e_1$, $A_2$ extends $e_2$ and $A_1 \prec_P A_2$

Similarly, an APS program P *cautiously* respects *o* if and only if

1. ∀ ⟨ $A_1$, $A_2$ ⟩ such that $A_1$ ∈ Answer Sets(P ∪ $C_1$), $A_2$, ∈ Answer Sets(P ∪ $C_2$), $A_1$ extends $e_1$, $A_2$ extends $e_2$ and $A_1 \prec_P A_2$

$ILP_{LOAS}^{context}$ task is of the form T = ⟨ B, $S_M$, $E_+$, $E_-$, $O^b$,$O^c$⟩ where $O^b$ and $O^c$ are brave and cautious orderings respectively, which are sets of ordering examples over set of positive partial interpretations $E^+$. A hypothesis H is an inductive solution of T if and only if

1. H ⊆ $S_M$ in $ILP_{LOAS}^{context}$
2. ∀⟨ e, C⟩ ∈ $E^+$, ∃A ∃ A ∈ Answer Sets (B ∪ C ∪ H) such that A extends e
3. ∀⟨ e, C⟩ ∈ $E^-$, ∄A ∃ A ∈ Answer Sets (B ∪ C ∪ H) such that A extends e

The two main advantages of adding contex-dependent are that it increases the efficiency of learning tasks, and more expressive structure of the background knowlege to particular examples. These features will be useful when a game agent is in two different environmets as discussed in Section **??**.

## 2.2 Reinforcement Learning (RL)

*Reinforcement learning (RL)* is a subfield of machine learning regarding how an agent behaves in an environment in order to maximise its total reward. As shown in Figure **??**, the agent interacts with an environment, and at each time step the agent takes an action and receives observation, which affects the environment state and the reward (or penalty) it receives as the action outcome. In this section, we briefly introduce the background in RL necessary for our research.

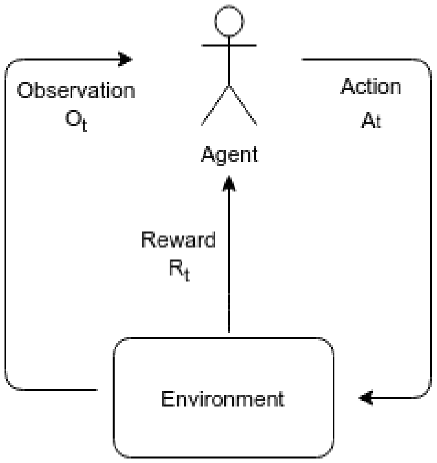

**Figure 2.1:** Agent and Environment

### 2.2.1 Markov Decision Process (MDP)

An agent interacts with an environment at a sequence of discrete time step, which is part of the sequential history of observations, actions and rewards. The sequential history is formalised as $H_t = O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t$. A *state* is a function of the history $S_t = f(H_t)$, which determines the next environment. A state $S_t$ is said to have *Markov property* if and only if $P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, ..., S_t]$. In other words, the probability of reaching $S_{t+1}$ depends only on $S_t$, which captures all the relevant information from the earilier history ([**?** ]). When an agent must make a sequence of decision, the sequential decision problem can be formalised using *Markov decision process (MDP)*. MDP formaly represents a fully observable environment of an agent for RL.
A MDP is of the form $\langle$ S, A, $T_a$, $R_a$, $\gamma$ $\rangle$ where:

- S is the set of finite states that is observable in the environment.
- A is the set of finite actions taken by the agent.
- $T_a(s, s')$ is a state transition in the form of probability matrix $Pr(S_{t+1} = s' \mid s_t = s, a_t = a)$, which is the probablity that action *a* in state *s* at time *t* will result in state *s′* at time *t+1*.
- R is a reward function $R_a(s, s') = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$, the expected immediate reward that action *a* in state *s* at time *t* will return.
- $\gamma$ is a discount factor $\gamma \in [0,1]$, which represents the preference of the agent for the present reward over future rewards.

### 2.2.2 Policies and Value Functions

*Value functions* estimate the expected return, or expected future rewarded, for a given action in a given state. The expected reward for an agent is dependent on the agent's action. The state value function $v_\pi(s)$ of an MDP under a policy $\pi$ is the expected return starting from state $s$, which is of the form:

$$v_\pi(s) = \mathbb{E}[G_t | S_t = s] \tag{2.2}$$

where $G_t = R_{t+1} + \gamma R_{t+2} + ... \gamma^{T-1} R_T$ , or the total discounted reward from $t$.
The optimal state-value function $v^*(s)$ maximises the value function over all policies in the MDP, which is of the form:

$$v^*(s) = \max_\pi v_\pi(s) \tag{2.3}$$

The optimal action-value function $q^*(s)$ maximises the action-value function over all policies in the MDP, which is of the form:

$$q^*(s, a) = \max_\pi q_\pi(s, a) \tag{2.4}$$

A solution to the sequential decision problem is called a *policy* $\pi$, a sequence of actions that leads to a solution. An optimal policy achieves the optimal value function (or action-value function), and it can be computed by maximising over the optimal value function (or action-value function).
TODO BELLMAN OPTIMALITY EQUATION
TODO Value iterations

### 2.2.3 Model-based and Model-free Reinforcement Learning

TODO delte dyna and focus more on model-based approach
A model M is a representation of an environment that an agent can used to understand how the environment should look like . Model-based learning is that the agent learns the model and plan a solution using the learnt model. Once the agent learns the model, the problem to be solved becomes a planning problem for a series of actions to achieve the agent's goal. Most of the reinforcement learning problems are model-free learning, where M is unknown and the agent learns to achieve the goal by solely interacting with the environment. Thus the agent knows only possible states and actions, and the transition state and reward probability functions are unknown.
The performance of model-based RL is limited to optimal policy given the model M. In other words, when the model is not a representation of the true MDP, the planning algorithms will not lead to the optimal policy, but a suboptimal policy.
One algorithm which combine both aspects of model-based and model-free learning to solve the issue of sub-optimality is called Dyna ([**?** ]), which is shown in Figure **??**.
Dyna learns a model from real experience and use the model to generate simulated experience to update the evaluation functions. This approach is more effective because the simulated experience is relatively easy to generate compared building up real experience, thus less iterations are required.

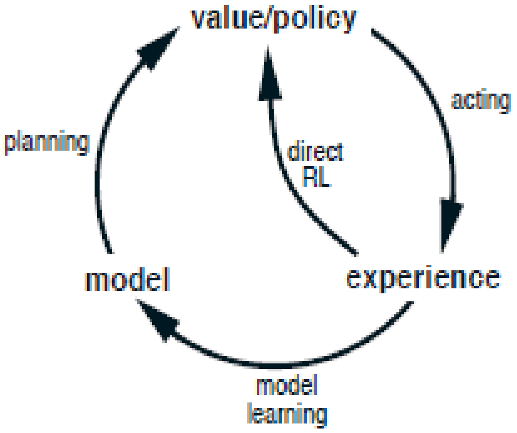

**Figure 2.2:** Relationships among learning, planning and acting

### 2.2.4 Temporal-Difference (TD) Learning

To solve a MDP, one of the approaches is called *Temporal-Difference (TD) Learning*. TD is an online model-free learning and learns directly from episodes of imcomplete experiences without a model of the environment. TD updates the estimate by using the estimates of value function by bootstrap, which is formalised as

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.5}$$

where $R_{t+1}$ + $\gamma$ $V(S_{t+1})$ is the target for TD update, which is biased estimated of $v_\pi$ $(S_t)$, and $\delta$ = $R_{t+1}$ + $\gamma$ $V(S_{t+1})$ - $V(S_t)$ is called TD error, which is the error in $V(S_t)$ available at time t+1. Since TD methods only needs to know the estimate of one step ahead and does not need the final outcome of the episodes, it can learn online after every time step. TD also works without the terminal state, which is the goal for an agent. TD(0) is proved to converge to $v_\pi$ in the table-based case (non-function approximation). However, because bootstraping updates an estimate for an estimate, some bias are inevitable.
*Q-learning* is off-policy TD learning defined in [**?** ], where the agent only knows about the possible states and actionns. The transition states and reward probability functions are unknown to the agent. It is of the form:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma max(a+t) Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{2.6}$$

where $\alpha$ is the learning rate, $\gamma$ is a discount rate between 0 and 1. The equation is used to update the state-action value function called Q function. The function Q(S,A) predicts the best action A in state S to maximise the total cumulative rewards.

$$Q(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|s_t, a_t] \tag{2.7}$$

Q-learning is guaranteed to converge to a optimal policy in a fininte tabulara representation. Paper Jaakkola et al. 1993
The optimal Q-function $Q^*$(s,a) is directly approximated by the learned action-value function Q.

**Algorithm 2** XXXX

```
procedure ILASP(RL) (B AND E)
    Initialise Q(s,a) arbitrarily
    Repeat (for each episode)
    Choose a from s using policy derived from Q (e.g, epsilon-greedy)
    Take action a, observe r, s′
```

Q-learning learns the value of its deterministic greedy policy from the experience and gradually converge to the optimal Q-function. It also explored following $\epsilon$-*greedy policy*, which is a stochastic greedy policy, but with the probability of $\epsilon$, the agent chooses an action randomly instea of the greedy action.

## 2.2.5 Function Approximation

Q-learning with tarbular method works when every state has Q(s,a). In case of very large MDPs, however, it may not be possible to represent all states with a lookup table. For example, robot arms has a continuous states in 3D dimentional space.
These problems motivate the use of function approximation, which estimates value function with function approximation. Not only it is represented in tabular form, but also in the form of a parameterized function with weight vector w $\in \mathbb{R}^{d}$ where $\mathbb{R}^{d}$ is XXX
Unlike Q-table, changing one weight updates the estimated value of not only one state, but many states, and this generalisation makes it more flexible to apply different scenarios that tabular approach could not be applied.
The reason we are introduing this function approximation is not because we will use it in our new algorithm, but for the benchmark that we compare our algorithm with.

### The Prediction Objective ($\overline{VE}$)

With function approximation, an update at one state changes many other states, and therefore the values of all states will not be exactly accurate, and there is a tradeoff among states as to which state we make it more accurate, while other might be less accurate.
The error in a state s is the squeare of the difference between the approximate value $\hat{v}$(s,w) and the true value $v_\pi$(s). The objective function can be defined by weighting it over the statespace by $\mu$, the *Mean Squared Value Error*, denoted $\overline{VE}$.

$$\overline{VE}(w) \doteq \sum_{s \in S} \mu(s) \big[ v_\pi(s) - \hat{v}(s,w) \big]^2. \tag{2.8}$$

### Stochastic gradient descent (SGD)

Stochastic gradient descent methods are commonly used to learn function approximation in value prediction, which works well for online reinforcement learning.

TODO EXPLAIN ONLINE VS OFFLINE LEARNING

$$w \doteq \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \tag{2.9}$$

and $\hat{v}$(s,w) is a differentiable function of w for all s $\in$ S.
minimize the $\overline{VE}$ on the observed examples. *Stochastic gradient-descent (SGD)* adjusts the weights vector by a fraction of alpha in the direction what will reduce the error on that example the most. Formally, it is defined as

$$\begin{aligned} w_{t+1} &\doteq w_t - \frac{1}{2}\alpha \bigtriangledown \big[v_\pi(S_t) - \hat{v}(S_t, w_t)\big]^2. \\ &= w_t - \alpha\big[v_\pi(S_t) - \hat{v}(S_t, w_t)\big] \bigtriangledown \hat{v}(S_t, w_t). \end{aligned} \tag{2.10}$$

where $\alpha$ is step-size,
The dradient of J(w) is defined as

$$\bigtriangledown_w J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix} \tag{2.11}$$

$$w_{t+1} \doteq w_t + \alpha\big[U_t - \hat{v}(S_t, w_t)\big]x(S_t) \tag{2.12}$$

### Linear Value Function Approximation

Formally,

$$\hat{q}(s,a) \approx q_\pi(s,a) \tag{2.13}$$

Represent state by a *feature vector*

$$x(S) = \begin{pmatrix} x_1(S) \\ \vdots \\ x_n(S) \end{pmatrix} \tag{2.14}$$

Use SGD updates with linear function approximation. The gradient of the approximate value function with respect to w is
Add proof here

$$\bigtriangledown \hat{v}(s,w) = x(s) \tag{2.15}$$

Thus the general SGD update defined in XX can be simplified to
Represent value function by a linear combination of features

$$\hat{v}(S,w) = x(S)^T w = \sum_{j=1}^{n} x_j(S) w_j \tag{2.16}$$

Objective function is The error in a state s is the square of the difference between the approximate value $\hat{v}$(S,w) and the true value v(S,w).

$$J(w) = \mathbb{E}_{\pi}\left[(v_{\pi}(S) - \hat{v}(S,w))^2\right] \tag{2.17}$$

Linear TD(0) is guaranteed to converge to clobal optimum

One disadvantage of the linear method is that it cannot express any relationship between features. For example, it cannot represent that feature *i* is useful only if feature *j* is not present.

Nevertheless, this approach is sufficient enough for our experiment, which will be described in Chapter XXX.

There are different linear methods to represents states as features, such as polynomials, fourier basis, or radial basis functions to name a few. Feature construction depends on a problem you are solving. In the next section, we introduce *Tile Coding* which will be used for our benchmark.

### Tile Coding TODO REFERENCE OF THIS METHOD

State set is represented as a continuous two-dimentionala space. If a state is within the space, then the value of the corrensponding feature is set to be 1 to indicate that the feature is present, while 0 indicates that the feature is absent. This way of representing the feature is called *binary feature*. *Coarse coding* represents a state with which binary features are present within the space. One area is associated with one weight w, and training at a state will affect the weight of all the areas overlapping that state. the approximate value function will be updated within at all states within the union of the areas, and a point that has more overlap will be more affected, as illustrated in Figure XX.

The size and shape of the areas will determine the degree of the generalisation. Large areaas will have more generalisation the change of the weight in that state will affect all other states within the intersection of the spaces.

The degree of overlap within a space will determined the degree of the generalisation.

The shape of the space also affect how it is generalised.

*Tile coding* is a type of coarse coding. *Tiling* is a partition of state space, and each element of the partition is called a *tile*.

In order to do coarse coding with tile coding, multiple tilings are required, each tiling is offset from one another by a fraction of a tile width.

As illustrated in Figure XXX, when a state occurs, several features with corresponding tiles become active,

Tile coding has computational advantage, since each component of tiling is binary value, XXXX.

a trained state will be generalised to other states if they are within any of the same tiles.

Similar to coarse coding, the size and shape of tiles will determine the degree of approximation.

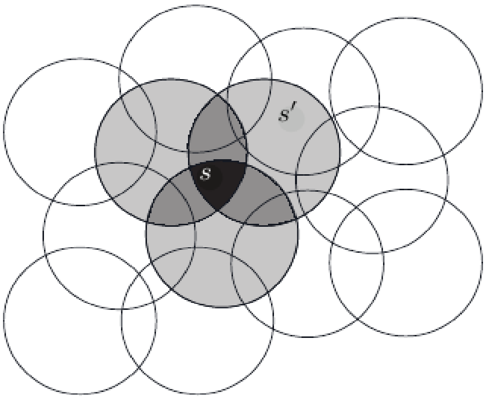

**Figure 2.3:** Relationships among learning, planning and acting Update this figure

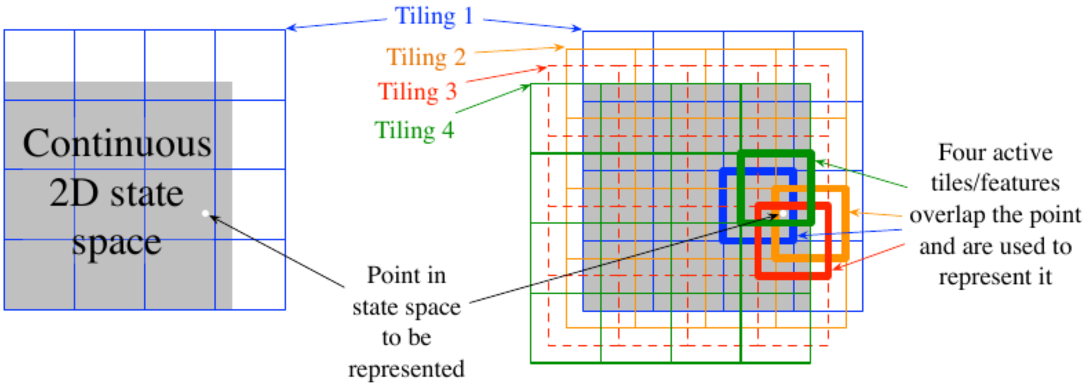
Tiling 1
Tiling 2
Tiling 3
Tiling 4
Continuous 2D state space
Point in state space to be represented
Four active tiles/features overlap the point and are used to represent it

**Figure 2.4:** Relationships among learning, planning and acting Update this figure

### 2.2.6 Transfer Learning

Transfer learning is a method that knowledge learnt in one or more tasks can be used to learn a new task better than if without the knowlege in the first task.
Transfer learning is an active research areas in machine learning, but not many have been done in RL. Since training tend to be time consuming and computational expensive, transfer learning allow the trained model to be applied in a different setting.
Transfer learning in RL is particularly important since most of the RL research have been done in a simulation or game scenarios, and training RL models in a real physical environment is more expensive to conduct.
Even in a virtual environments like games, the transfer learning between different tasks will greatly will have a big impact on potential applications.
This will also speed up learning
Transfer learning in ILP domain have been proved to be successful in many fields, Since this project is combining ILP into RL senarios, this has a potential for extending this particular research.

We conducted experiements on transfer learning capabilities, which we describe in XXX.
One of the purposes of transfer learning is so that the agent requires less time to learn a new task with the help of what was learn in previous tasks.
Another goal would be to measure how effectively the agent reuses its knoledge in a new task. In this case the performan of learningon the first task is usally not measured.
There are many different matrices used to measure the performance of the transfer learning. Five common matrics are defined in XX as follow.
TODO source task selection

- Jumpstart
- Asymptotic Performance

Since each matric measures different aspect of transfer learning, using multiple metrics would provide more comprehensive views of the performance of an RL algorithm.

$$r = \frac{Areaundercurvewithtransfer - areaundercurvewithouttransfer}{areaundercurvewithouttransfer} \tag{2.18}$$

REFERENCE

# Chapter 3

# Framework

The overall pipeline is shown in Figure **??**. By interacting with the environment, an agent accumulates state transition experiences as positive examples, which is used by ILASP to learn and improve hypothesis. The agent also records surrounding information it has seen as background knowledge, which is used to make a plan together with the hypothesis that ILASP learns by solving an answer set program. Mechanisms of each step is explain in details in the following sections.

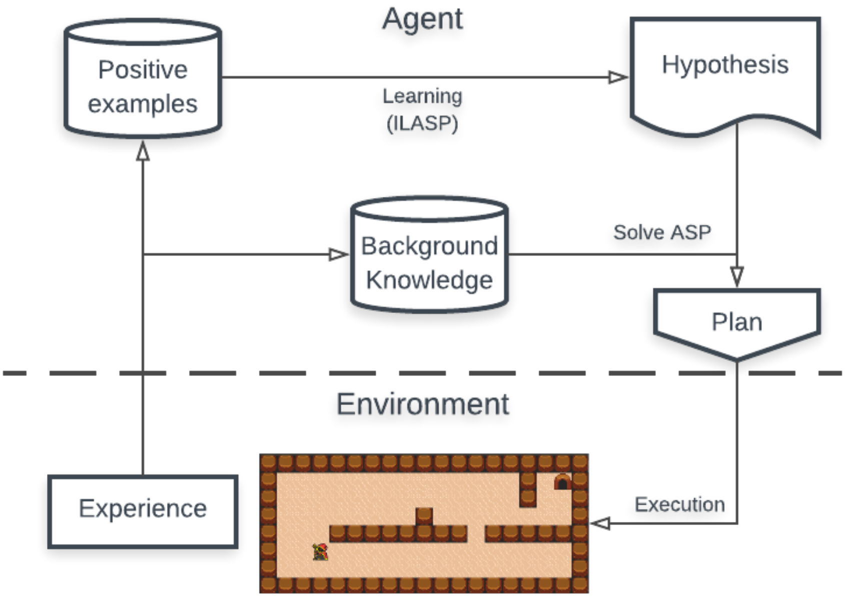

**Figure 3.1:** Reinforcement learning pipeline using ILASP. ILASP learns to generate a modelof the environment, or hypothesis, and updates it based on the interaction with the environment.

## 3.1 Experience Accumulation

Draw an illustration of the difference between exploration and las part
Related these with Definition of ILASP
The first step is to accumulate experience by interacting with the environment. Similar to tranditional RL agent, an agent explores an environment randomly until it reaches the goal once. Every time the agent takes an action during the exploration phase, these experiences need to be recorded in two different forms: *state transition experience* and *environment experience*.

### 3.1.1 State transition experience

*State transition experience* contain information about how the state transition at each time step. It is the same as $E^+$ in ILASP, especially positive examples for ILASP in ASP syntax, which is of the form:

$$\begin{aligned}&\#\mathsf{pos}(\{\mathsf{state\_after((X2,Y2))}\},\\ &\qquad \{\mathsf{all\ other\ state\_after\ that\ did\ not\ happen}\},\\ &\qquad \{\mathsf{state\_before((X1,Y1)).\ action(A).\ surrounding\ information}\}).\end{aligned} \tag{3.1}$$

where,

- inclusions contain one state_after((X2,Y2)), which represents the position of the agent in x and y axis after an action is taken
- exclusions contain all other state_after((X,Y)) that did not occur
- context example include state_before((X1,Y1)), which represents the position of the agent in x and y axis before an action is taken, action(A) is the action the agent has taken, and surrounding information, such as walls, if any.

These experience need to be expressed in ASP form in order to execute the inductive learning in ILASP. The input used in ILASP is state transitions, rewards and an action of the agent,
exclusions are determined by the state that did not happen. The inclusions and exclusions are determined by the following algorithms

$$\forall s \in \mathsf{State,\ agent\ is\ at\ s,\ ASP\ of\ H\ does\ not\ contains\ s\ as\ state\_after,\ add\ s} \rightarrow \mathsf{INC}.$$
$$\forall s \in \mathsf{State,\ agent\ is\ not\ at\ s,\ ASP\ of\ H\ contains\ s\ as\ state\_after,\ add\ s} \rightarrow \mathsf{EXC}.$$

Where INC and EXC are inclusions and exclusions respectively. ASP is Answer Set Program P and H is the current hypothesis. context example are

- the state that the agent was before taking action (represented as *state_before(x,y)*)
- an action that the agent takes (representend as *action(a)*)
- surrounding information of *state_before(x,y)*, such as walls.

Using these positive examples, the agent is able to learn and improve hypothesis as it explore the environment and encounters new scenarios.

**example 3.1.1.** (Positive examples).
Suppose an agent takes an action ”up” to move from (1,3) to (1,4) cell. All other alternative states that the agent could have ended up by taking different actions (down, right, and left) are in the exclusions. Finally surrounding walls information are in the context.

```
#pos({state_after((1,3))},
        {state_after((2,4)),state_after((1,5)),state_after((0,4)),state_after((1,4))},
        {state_before((1,4)). action(up). wall((1, 5)). wall((0, 4)).})
```

This example will be used to learn how to move up as one of the agent’s hypotheses. Another example is

```
#pos({state_after((1,3)),
        {state_after((2,3)),state_after((1,4)),state_after((0,3)),state_after((1,2))}
        {state_before((1,3)). action(up). wall((2,3)). wall((0,3)). wall((1,2)).}).
```

where the agent tried to move up, from (1,3) to (1,2), but ended up in the same cell at (1,3). This is because there is a wall at (1,2), and the agent learns in order to move an above cell, there must not be a wall in the above cell.

### 3.1.2 Environment experience

While the agent explores in the environment, it also remembers all the surrounding information as background knowledge, which will be stored in different repository, and are later used to generate a sequence of actions plan using H.
These experience correspond to B in ILASP definition.
This does not include state transition experience, as these state_before and action takens are different at every timestep. Backgroud knowledge will be
In static environment (e.g no moving enermy), environment information remain the same across time, and thus it will be beneficial to remember. This principle is different from most RL methods, as they are model-free learning. As described in XX,, model-based learning converge to optimal policy more efficiently.
In a simple maze, these could be all wall position that the agent has seen so far, which can be wall((1, 5)). which represents the location of the wall. Another example could be a location of a teleportation if the agent sees it.
These environment experiences are part of context examples in the positive examples.

## 3.2 Inductive Learning

Throughout the learning process, the agent accumulates positive examples and learn hypothesis H.
Our learning task is XXX.
In order to execute ILASP, the following definitions are supplied as well as the positive examples,

### 3.2.1 Search Space

In order to execute ILASP, a *search space* of possible hypotheses is required, which is defined using a *language bias*.

```
#modeh(state_after(var(cell))).
#modeb(1, adjacent(const(action), var(cell), var(cell))).
#modeb(1, state_before(var(cell)), (positive)).                (3.2)
#modeb(1, action(const(action)),(positive)).
#modeb(1, wall(var(cell))).
```

Without these in the form of mode bias, the search space for ILASP will be empty.

```
cell((0..7, 0..6)).
adjacent(right, (X+1,Y),(X,Y)) :- cell((X,Y)), cell((X+1,Y)).
adjacent(left,(X,Y), (X+1,Y)) :- cell((X,Y)), cell((X+1,Y)).   (3.3)
adjacent(down, (X,Y+1),(X,Y)) :- cell((X,Y)), cell((X,Y+1)).
adjacent(up, (X,Y), (X,Y+1)) :- cell((X,Y)), cell((X,Y+1)).
```

where *var(t)* and *const(t)* are a placeholder for variable and constant terms of type t respectively.
*const(t)* must be specified as #constant(t,c), where t is a type and c is a constant term. In our environment, action is specified as constant since ILASP should learn different hypothesis for each action.

```
#constant(action, right).
#constant(action, left).
                                (3.4)
#constant(action, down).
#constant(action, up).
```

As we describe in XXX, the search space increases in propotion to the complexity of learning tasks, which slows down the learning process. For example, the search space in this particular setting is in XX.

#max_penalty defines the maximum size of the hypothesis, by default it is 15. Increasing #max_penalty allows ILASP to learn longer hypothesis in expense of longer computation. #max_penalty(50).
Together with the above defition as well as accumulated positive examples, ILASP is able to learn an hypothesis. The quality of H depends on the experiences for the agent. For example, In the early phase of learning, the agent does not have many examples, and learns an hypothesis that may not be insightfull. For example, if the agent has only one positive example,
Next, the scope of *cell* are defined, as cell((0..X, 0..Y)), where X and Y are size of width and height respectively.
Finally, since our learning task is the rule of the game, which involve state transition, it needs to know how it means to be ”being next to XX”, Therefore the following assumptions are provided as background knowledge.

```
state_after(V0) :- adjacent(right, V0, V1), state_before(V1), action(right), not wall(V0).
state_after(V0) :- adjacent(left, V0, V1), state_before(V1), action(left), not wall(V0).
state_after(V0) :- adjacent(down, V0, V1), state_before(V1), action(down), not wall(V0).
state_after(V0) :- adjacent(up, V0, V1), state_before(V1), action(up), not wall(V0).
                                                                                   (3.5)
```

This definition itself could be learnt by setting another learning tasks, and it is a potential learning problem. However, we focus on learning task of the rule of the game in this paper.
The full details for ILASP learning tasks is described in Appendix XXX. Positive excludes the possibility of negation as a failure in order to reduce the search space.
Future research will relax these assumptions and attemp to learn more general hypothesis, e.g learning adjacent defintion.
These learnt H will be used to generate a plan in the abduction phase.
After executing the plan, the agent will have more positive examples, which will be used to improve the quality of H.
The learnt hypothesis is XXX
This hypothesis, for example, does not explain how to move ”down”. In order to learn how to move ”down”, it needs an positive example of moving up.
later on H improving as we collect more examples as well as background knowledge.

## 3.3 Generate a plan

Once the agent find the goal once, we can generate a plan using the current hypothesis by solving Answer Set Program.
If the hypotheses were not accurate, clingo might not generate all the actions leading to the goals.
The syntax of ASP is different from ILASP phase, because we need to include time sequence when solving ASP. In ILASP, it is only state_before and staet_after, but in

plan generation, there will be more than one state transition.

$$\begin{aligned}&1\{\mathsf{action(down,T);\ action(up,T);\ action(right,T);\ action(left,T);\ action(non,T)}\}1\\&\ \mathsf{:\text{-}\ time(T),\ not\ finished(T).}\end{aligned} \tag{3.6}$$

This choice rule states that action must be one of four actions (defined maximum and minimum numbers in 1), T is the time step at which the agent takes each action, unless *finished(T)* is grounded.
*finished(T)* is associated with goal definition, and it is defined as:

$$\begin{aligned}&\mathsf{finished(T)\text{:-}\ goal(T2),\ time(T),\ T} \geq \mathsf{T2.}\\&\mathsf{goal(T)\text{:-}\ state\_at((5,\ 1),\ T),\ not\ finished(T\text{-}1).}\\&\mathsf{goalMet\text{:-}\ goal(T).}\\&\mathsf{:\text{-}\ not\ goalMet.}\end{aligned} \tag{3.7}$$

## 3.4 Plan execution

the plan generated by clingo is a set of states and actions.
states are of the form state_at((X,Y),T), where X and Y represent x-axi and y-axi in a maze respectively, T represents a time that the agent is at this particular X,Y cell.
action(A,T) tells which action the agent should take at each time. By following the actions, the agent should collect both predicted state that the agent will end up, and the observed state that the agent actually end up. If there is a difference between these two, either B or H do not correctly represent the model of the true environment, so needs to be improved.
When the agent encounters a new environment (e.g a new wall), this new information will be added to its background, which will be used to improved the hypothesis next time ILASP gets executed.
For example,

$$\begin{aligned}&\mathsf{state\_at((1,1),1),\ action(right,1)}\\&\mathsf{state\_at((2,1),2),\ action(right,2)}\\&\mathsf{state\_at((3,1),3),\ action(right,3)}\\&\mathsf{state\_at((4,1),4),\ action(right,4)}\\&\mathsf{state\_at((5,1),5),} \cdots\end{aligned}$$

At the start of the learning, H is usually not correct or too general, using this H will generate lots of answer sets that are not useful for the planning. These examples will be collected and included as exclusions of a new positive example.
To avoid the agent from being stuck in a sub-optimal plan, the agent deliberately discards the plan and takes an random action with a probability of epsilon (which is less than 1) TODO define this mathematically. When the agent deviates from the planning, it often discovers new information, which will be added to B. Exploration

is necessary to make sure that the agent might discovers a shorter path than the current plan, which will be demonstrated in the experiment.

Define them here

Our algorithm works by

It buids the model of the environment by improving two internal concepts: hypothesis H and background knowledge B.

In the further research, we could experiment with a more sophisticated exploration strategy, such as XXX and YYY.

This is formally defined in Algorithm.

**Algorithm 4** XXXX

```
procedure ILASP(RL) (B AND E)
    while True do
        H (inductive solutions) ← run ILASP(T)
        plan(actions, states) answer sets ← AS(B, H)
        while actions in P do
            observed state ← run clingo(T)
            if observed state ≠ predicted state then
                H ← run ILASP(T)
```

Everytime the agent executes an action by following the plan, it checks whether the observed state is that is expected.

If there is a difference between the two, either

B is incorrect

H is not sophisticated enough,

If that is the case, the agent runs ILASP again using more positive examples it collected during the plan execution.

## 3.5 Exploration

our algorithm kicks in once the agent reaches a goal once. However it is likely that the agent has not seen all the environment and therefore is likely to be in a sub-optimal plan. Therefore, similar to RL algorithm, the agent also has to explore a new state. There are a number of exploration strategy in RL (such as Boltzman approach, Count-based and Optimistic Initial value TODO REFERENCE). One of the most commonly used strategy is $\epsilon$-greedy strategy. As described in Chapter XXX, the agent takes an random action

This strategy may not be appropriate in cases there safety is a priority (since it is random action.) It is simple to implement. In the case of our algorithm, the agent discard the plan from the abduction with a probably of epsilon and takes a random action in order to avoid getting stuck in a sub-optimal path. When the agent takes an random action and move into a new state, the agents creates a new plan from the new state and continue to move forward.

This exploration point will be highlighted in Experiment XXX.

Epsilon needs to be larger than Q-learnig because

# Chapter 4

# Evaluation

## 4.1 Evaluation Metrics

The two main measurements for the performance of our new architecture are learning efficiency and transfer learning capability. The learning efficiencies are measured in two different ways. First, the algorithm performance is compared with exisitng RL algorithms in terms of convergence rate, which is measured in terms of number of episodes that the agent needs to get to an optimal policy. Second, ILASP learning is measured in each episodes and time steps to see how the new algorithm refines its internal hypothesis over time. Other common measurements in RL are XXX, XXX, but our primary goal of this research is data-efficient learning.

### 4.1.1 Experiment Platform

We use the Video Game Definition Language (VGDL), which is a high-level description language for 2D video games providing a platform for computational intelligence research ([**?** ]). The VGDL allows users to easily craft their own environments, which makes us possible to do various experiments without relying on a default environment. The VGDL platform provides an interface with OpenAI Gym ([**?** ]), which is a commonly used benchmark platform. The base game is a simple maze as shown in Figure **??**. There are 3 different types of cells: a goal cell, walls and paths. The agent can take 4 different actions: up, down, right and left. The environment is not known to the agent in advance, and it attempts to find the goal by exploring the environment.

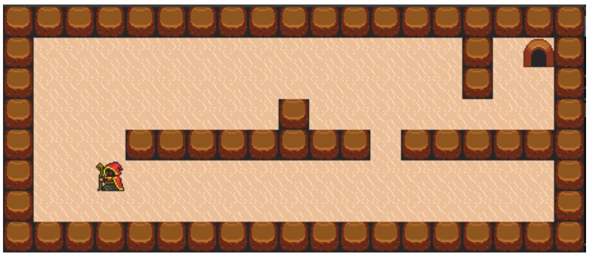

**Figure 4.1:** VGDL game example

### 4.1.2 Benchmark

We use two existing reinforcement learning methods as benchmarks: Q-learning and tile-coding. Q-learning is most widely used technique for reinforcement learning. Given our environment is deterministic and discrete, this method works well in our environment. Another benchmark is tile coding, which is a type of linear function approximation techniques described in Chapter XX. The reason for using another extra benchmark is that the comparision with q-learning might not be a fair comparision, since our algorithm has one extra assumptions: the agent knows surrounding information (whether there are walls in adjacent cells), which is not a common assumption for Q-learning. Thus we incorporate the same surrounding information as features, and update the weights of each feature as a learning. We compare the performance of our algorithm with these two methods.

### 4.1.3 Parameters

In all experiments, the agent receives -1 in any states except the goal state, where it gains a reward of 10. All the matrices that are used in the experiments are summarised in Table **??**.

| Parameter | My algorithm | Benchmark |
|---|---|---|
| The number of episode | 100 | 100 |
| Time step per episode | 250 | 250 |
| The number of experiments | 30 | 30 |
| Alpha | N/A | 0.5 |
| Epsilon | 0.4 | 0.1 |

**Table 4.1:** Parameters of the initial and optimised fully connected network.

Epsilon for our algorithm should be higher, since the agent follows the generated plan, whereas benchmark algorithms update value function with the degree of alpha. We conducted several experiments using different environments to highlight each aspect of the algorithm.
Since the performance of the agent is affected by the randomness of the exploration, and our algorithm is highly dependent on how quickly the agent finds the goal, each experiment is conducted 30 times and the perform is averaged across the experiments. At each episode, we also measure the performance without exploration to see the pure optimal policy.

## 4.2 Experiment Results

### 4.2.1 Experiment1

The purpose of the first experiment is how the algorithm learns the model of the environment, or hypothesis in ILASP. The environment are defined as a simple maze where the goal is located the right uppper corner as shown in Figure **??**.

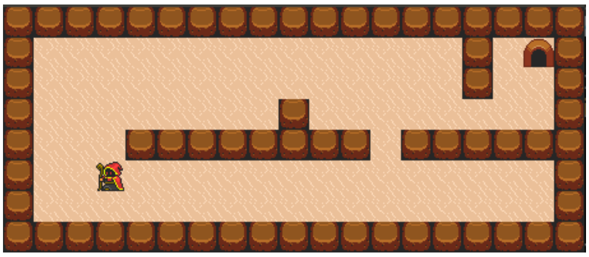

**Figure 4.2:** Enviroment for experiment 1

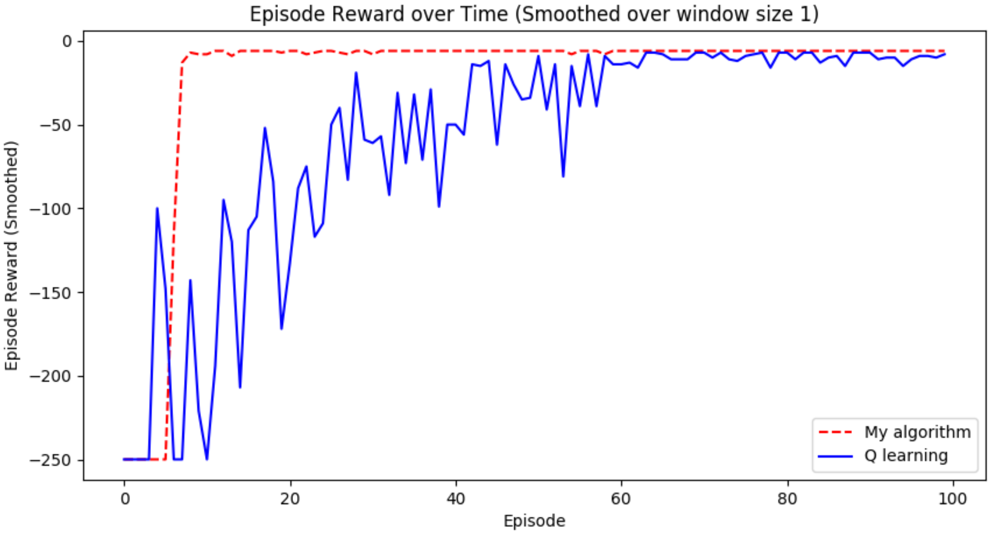

**Figure 4.3:** Comparison of training performance between my algorithm and Q-learning

Figure **??** shows the traning performance between our algorithm and Q-learning. The convergence rate of our algorithm is faster than Q-learnig. This is because unlike Q-learning where the value function is updated with the rate of alpha, ILASP gradually build the model of the environment and use the background knowledge to accurately plan. The same trend is also shown in Figure **??**, where we measure only the performance of the policy without random exploration.

Overall this results shows that our algorithm converges to the optimal policy faster than transitional RL algorithm in a simple scenarios, achieving more data-efficient learning.

The second improvement of our algorithm is transparency. Learnt hypotheses are shown in **??**, which is the rule of the game and easy to understand for human users. Since the learnt hypothesis is a general concept, which can be used in a different environmet. This transfer learning capability is also described in Experiement XX.

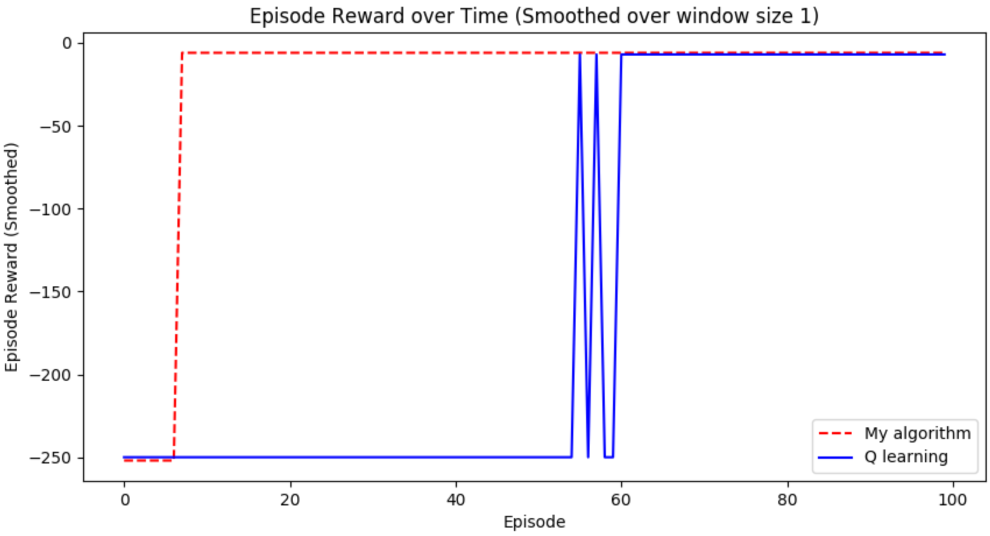

**Figure 4.4:** Comparison of test performance between my algorithm and Q-learning

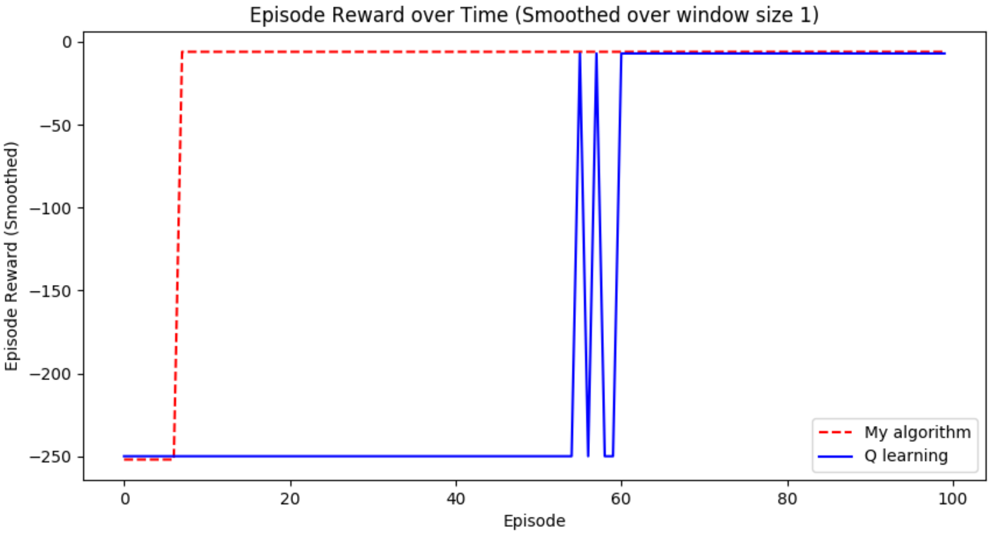

**Figure 4.5:** PLACEHOLDER FOR ILASP

ILASP learns the final hypothesis at time XX in episode 0.

state_after(V1) :- adjacent(right, V0, V1), state_before(V1), action(right), wall(V0).
state_after(V0) :- adjacent(right, V0, V1), state_before(V0), action(left), wall(V1).
state_after(V1) :- adjacent(down, V0, V1), state_before(V1), action(down), wall(V0).
state_after(V1) :- adjacent(up, V0, V1), state_before(V1), action(up), wall(V0).
state_after(V0) :- adjacent(right, V0, V1), state_before(V1), action(right), not wall(V0).
state_after(V0) :- adjacent(left, V0, V1), state_before(V1), action(left), not wall(V0).
state_after(V0) :- adjacent(down, V0, V1), state_before(V1), action(down), not wall(V0).
state_after(V0) :- adjacent(up, V0, V1), state_before(V1), action(up), not wall(V0).

(4.1)

### 4.2.2 Experiment2

The first experiment might not be a fair comparision between our algorithm and Q-learning, since our algorithm has extra information about the surrounding information. In order to have the same assumptions, we use function approximation for Q-learning. Linear function approximation.
In this experiment, the performance of our algorithm remains the same as Experiment 1, but we use a different benchmark.

### 4.2.3 Experiment3

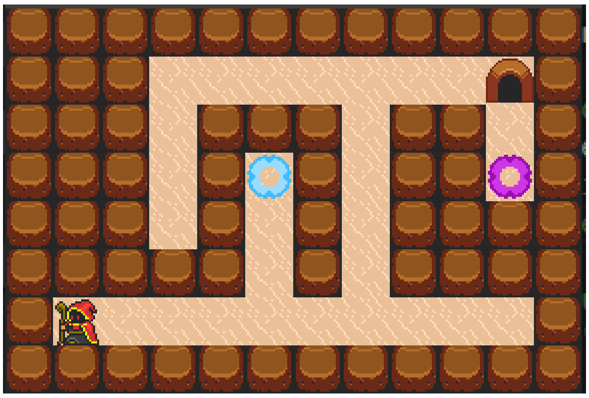

**Figure 4.6:** Enviroment for experiment 3

This experiment is conducted to see if the agent find a optimal path of using a teleport. There are two ways to reach the goal: using a normal path or using a telport. The environment is designed such that using a teleport is a shorter path and therefore gives higher total reward.

Compared to the first experiment, additional search space and concept are supplied as follow:

```
#modeb(1, link_start(var(cell)), (positive)).
#modeb(1, link_dest(var(cell)), (positive)).
```

Where an one way teleport link is added to the environment: link_start takes the agent to link_dest, but link_dest does not take the agent back to link_start. The allows ILASP to learn additional hypothesis. The full learning task for this experiment is in Appendix XX.

Once the agent steps onto a state where link_start is located, it gets two positive experiences. In this game environment, the agent moves two cells in one time step instead of one cell per time step.

Also link_start and link_dest need to be stored in background knowledge rather than contex, because ILASP needs to learn different hypothesis for link and non-link case. link locations need to be available for all positive examples so that ILASP correctly learn non-link, which is shown in Figure XX below.

The training performance shown in XX, which converges faster than XX.

ILASP learns the final hypothesis at time XX in episode 0. To highlight the learning the new concept of teleport link, Figure XX is an intermediate incomplete hypothesis

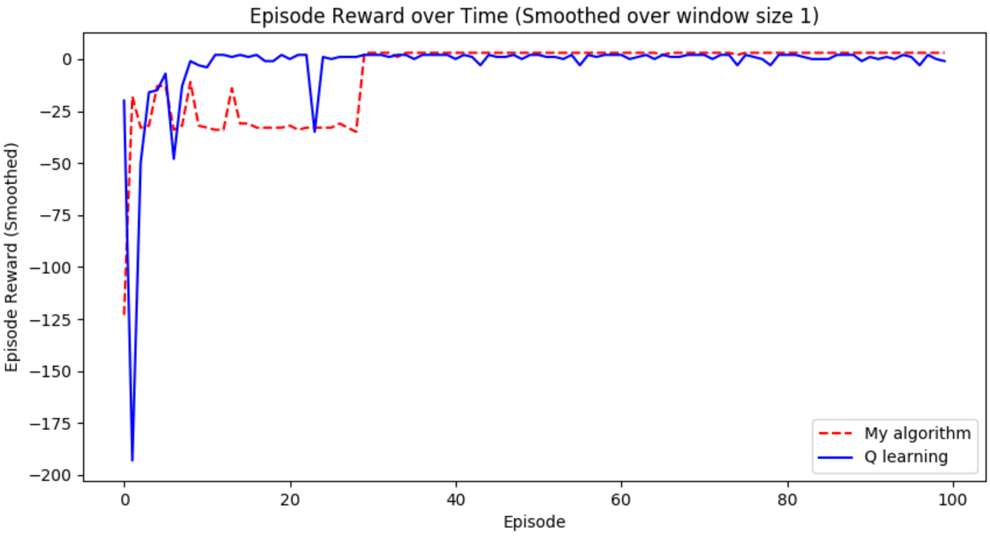

**Figure 4.7:** Comparison of training performance between my algorithm and Q-learning

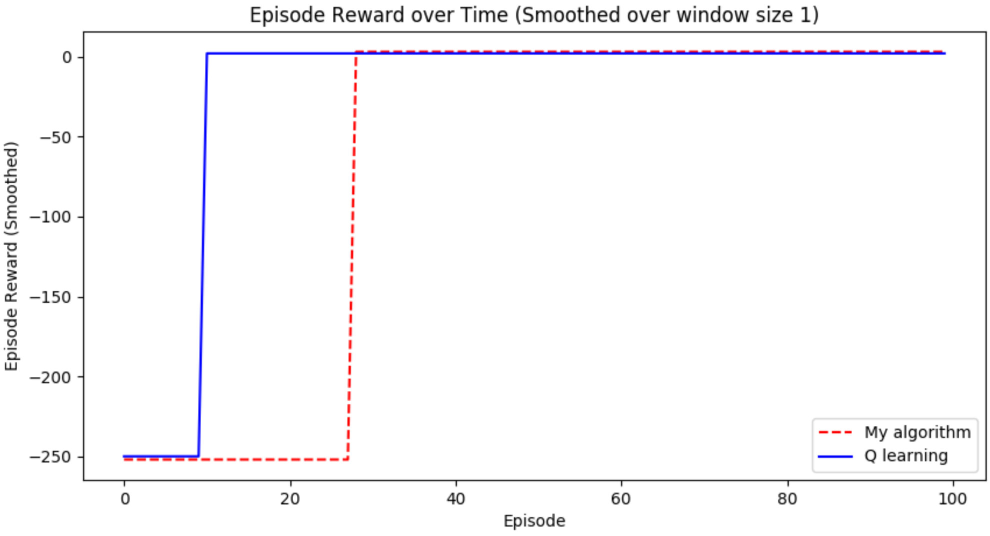

**Figure 4.8:** Comparison of test performance between my algorithm and Q-learning

learnt by ILASP.

```
state_after(V1) :- link_dest(V1).
state_after(V0) :- link_dest(V0), state_before(V0), action(right).
state_after(V1) :- adjacent(left, V0, V1), state_before(V0), action(right), not wall(V1).
state_after(V0) :- adjacent(left, V0, V1), state_before(V1), action(left), not wall(V0).
state_after(V1) :- adjacent(up, V0, V1), state_before(V0), action(down), not wall(V1).
state_after(V0) :- adjacent(up, V0, V1), state_before(V1), action(up), not wall(V0).
state_after(V1) :- adjacent(left, V0, V1), state_before(V1), action(left), wall(V0).
state_after(V1) :- adjacent(down, V0, V1), state_before(V1), action(down), wall(V0).
state_after(V1) :- adjacent(up, V0, V1), state_before(V1), action(up), wall(V0).
```

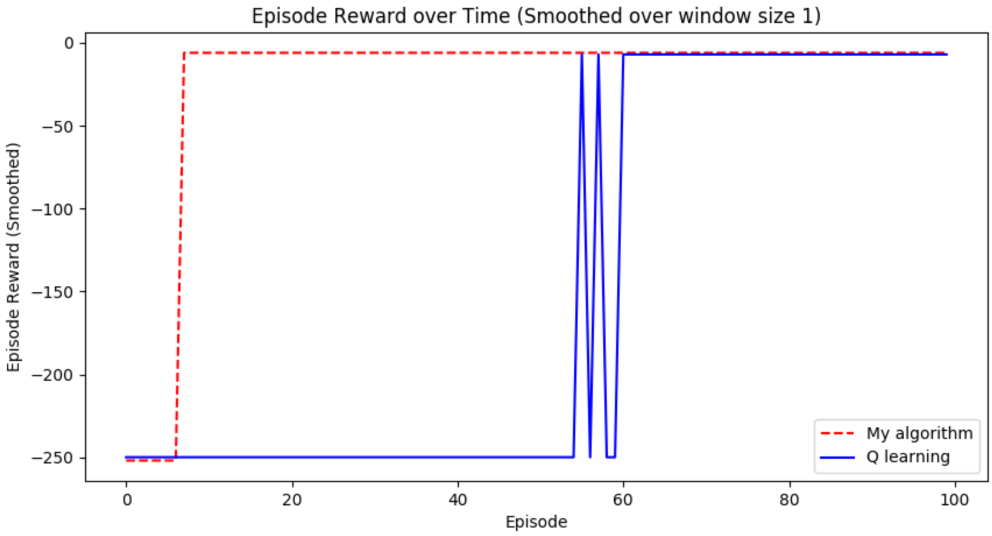

**Figure 4.9:** PLACEHOLDER FOR ILASP

These hypotheses are generated just after the agent steps onto the link. However, the first hypothesis says whenever link_dest is available state_after is true. Since link_dest is available in background knowledge rather than context, when solving for answer sets to generate a plan, it generates incorrect state_after at every time step. However, as shown in Algorithms XX, these generated state_after are all incorrect and therefore will be added to exclusions of the next positive examples. These exclusions will later refines hypotheses and results in Figure XX, the final complete hypotheses.
Learnt hypotheses are as follow:

```
state_after(V1) :- link_start(V0), link_dest(V1), state_before(V0).
state_after(V0) :- link_dest(V0), state_before(V0), action(right).
state_after(V1) :- adjacent(left, V0, V1), state_before(V0), action(right), not wall(V1).
state_after(V0) :- adjacent(left, V0, V1), state_before(V1), action(left), not wall(V0).
state_after(V1) :- adjacent(up, V0, V1), state_before(V0), action(down), not wall(V1).
state_after(V0) :- adjacent(up, V0, V1), state_before(V1), action(up), not wall(V0).
state_after(V1) :- adjacent(left, V0, V1), state_before(V1), action(left), wall(V0).
state_after(V1) :- adjacent(down, V0, V1), state_before(V1), action(down), wall(V0).
state_after(V1) :- adjacent(up, V0, V1), state_before(V1), action(up), wall(V0).
```

Compared the Experiment 1, there are two new hypotheses due to the presence of the telport links. These learnt hypotheses are also applicables to an environment where there is not link, such as a game in experiment 1. In this case, the first two hypotheses in Figure XX are never be used since the body predicates are never be true.

### 4.2.4 Experiment4

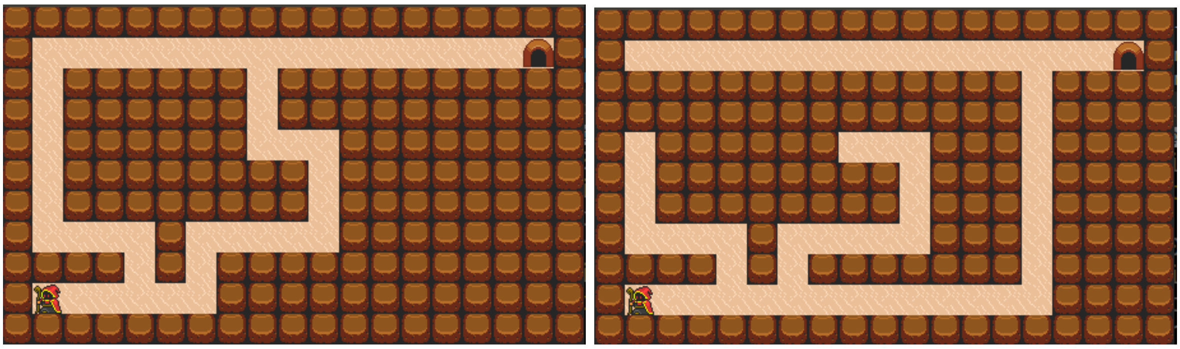

**Figure 4.10:** Before (left) and after (right) transfer learning

Finally, we investigated the potentials of transfer learning betweeen similar environments.

The goal position is the same as in the first game, but the routes to the goal are different.

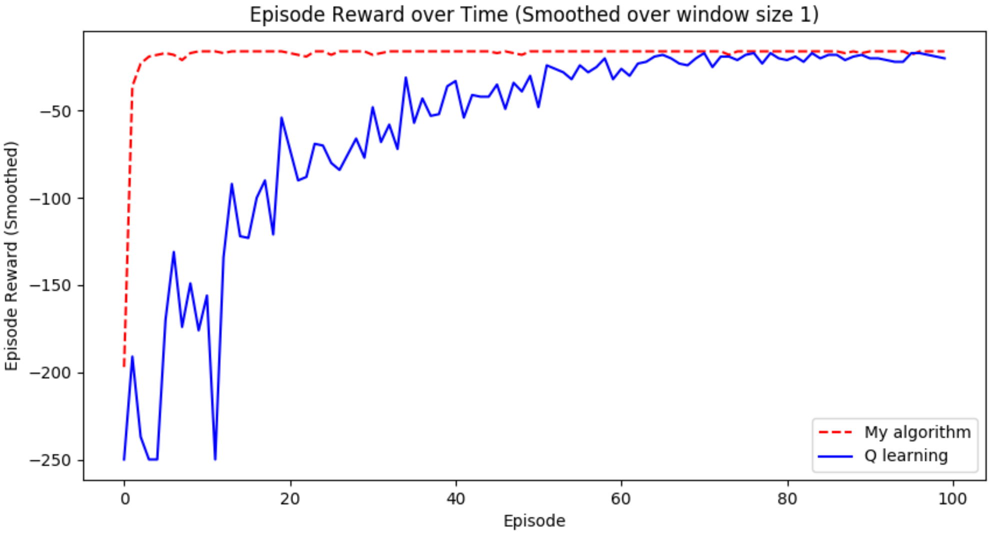

**Figure 4.11:** Comparison of training performance between my algorithm and Q-learning (before transfer)

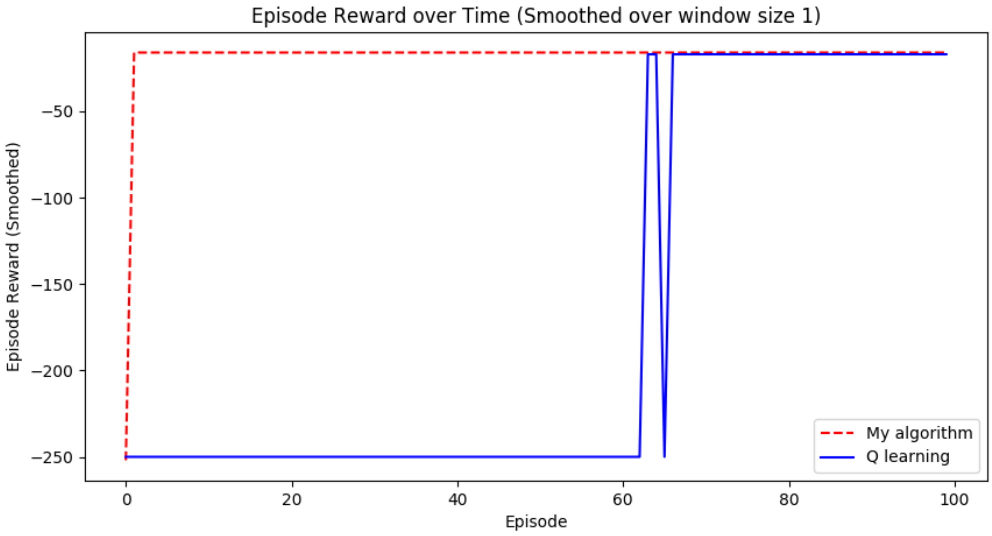

**Figure 4.12:** Comparison of test performance between my algorithm and Q-learning (before transfer)

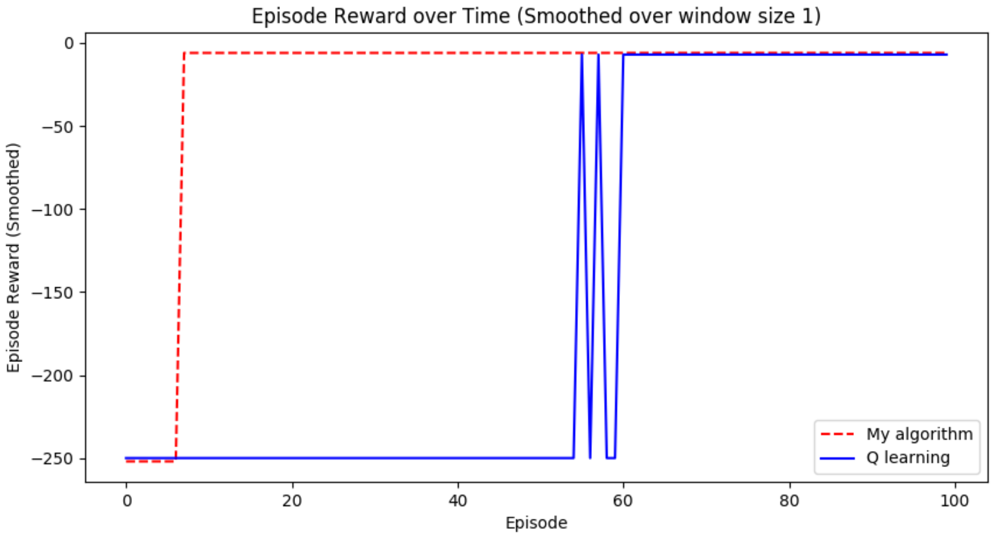

**Figure 4.13:** PLACEHOLDER FOR ILASP

These are the hypotheses we are transferring to a new environment.

```
state_after(V0) :- adjacent(right, V0, V1), state_before(V1), action(right), not wall(V0).
state_after(V0) :- adjacent(left, V0, V1), state_before(V1), action(left), not wall(V0).
state_after(V1) :- adjacent(down, V0, V1), state_before(V0), action(up), not wall(V1).
state_after(V0) :- adjacent(down, V0, V1), state_before(V1), action(down), not wall(V0).
state_after(V1) :- adjacent(right, V0, V1), state_before(V1), action(right), wall(V0).
state_after(V1) :- adjacent(left, V0, V1), state_before(V1), action(left), wall(V0).
state_after(V0) :- adjacent(up, V0, V1), state_before(V0), action(down), wall(V1).
state_after(V1) :- adjacent(up, V0, V1), state_before(V1), action(up), wall(V0).
```

### Transfer learning

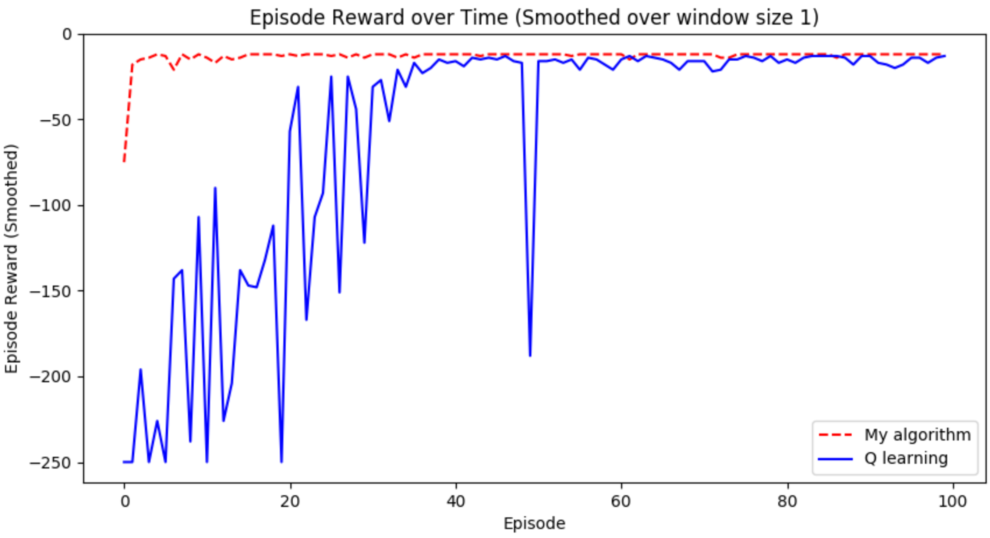

**Figure 4.14:** Comparison of training performance between my algorithm and Q-learning (after transfer)

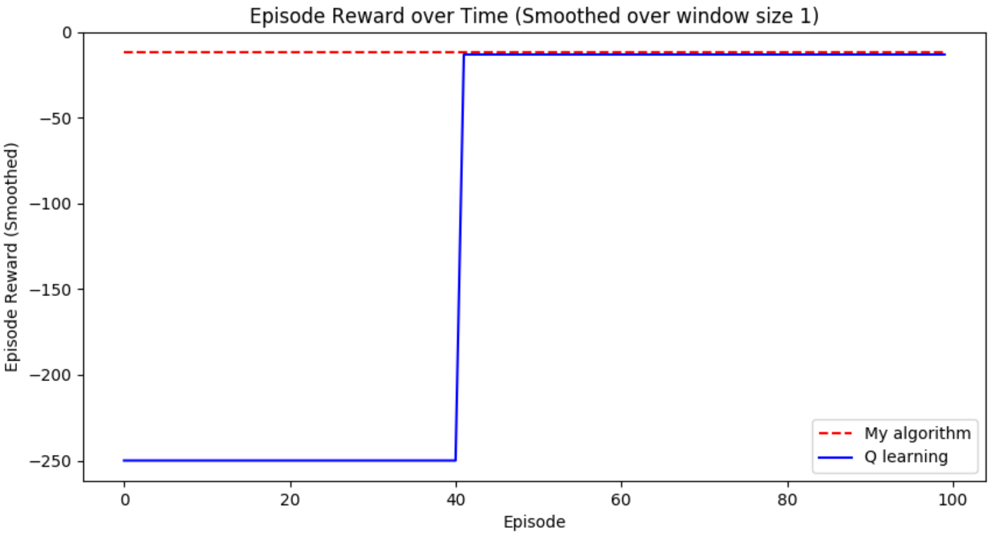

**Figure 4.15:** Comparison of test performance between my algorithm and Q-learning (after transfer)

These are the hypotheses we are transferring to a new environment.

### 4.2.5 Experiment5

TODO Transfer learning for new H

# Chapter 5

# Related Work

In this section, I summarise recent studies related to symbolic (deep) reinforcement learning.

[? ] introduced Deep Symbolic Reinforcement Learning (DSRL), a proof of concept for incorporating symbolic front end as a means of converting low-dimensional symbolic representation into spatio-temporal representations, which will be the state transitions input of reinforcement learning. DSRL extracts features using convolutional neural newtworks (CNNs) [? ] and an autoencoder, which are transformed into symbolic representations for relevant object types and positions of the objects. These symbolic representations represent abstract state-space, which are the inputs for the Q-learning algorithm to learn a policy on this particular state-space. DSRL was shown to outperform DRL in stochastic variant environments. However, there are a number of drawbacks to this approach. First, the extraction of the individual objects was done by manually defined threhold of feature activation values, given that the games were geometrically simple. Thus this approach would not scale in geometrically complex games. Second, using deep neural network front-end might also cause a problem. As demonstrated in [? ], a single irrelevant pixel could dramatically influence the state through the change in CNNs. In addition, while proposed method successfully used symbolic representations to achieve more data-efficient learning, there is still the potential to apply symbolic learning to those symbolic representations to further improve the learning efficiency, which is what we attemp to do in this paper. [? ] further explored this symbolic abstraction approach by incorporating the relative position of each object with respect to every other object rather than absolute object position. They also assign priority to each Q-value function based on the relative distance of objects from an agent.

[? ] added relational reinforcement learning, a classical subfield of research aiming to combining reinforcement learning with relational learning or Inductive Logic Programming, which added more abstract planning on top of DSRL approach. The new mode was then applied to much more complicate game environment than that used by [? ]. This idea of adding planning capability align with our approach of using ILP to improve a RL agent. We explore how to effectively learn the model of the environment and effectively use it to facilitate data-efficient learning and transfer learning capability.

Another approach for using symbolic reinforcement learning is storing heuristics expressed by knowledge bases [[**?** ]). An agent lerans the concept of *Hierarchical Knowledge Bases (HKBs)* (which is defined in more details in [**?** ] and [**?** ]] at every iteration of training, which contain multiple rules (state-action pairs). The agent then is able to decide itself when it should exploit the heuristic rather than the state-action pairs of the RL using *Strategic Depth*. This approach effectively uses the heuristic knowledge bases, which acts as a sym-symbolic model of the game.

Another field related to our research is the combining of ASP and RL. The original concept of combining ASP and RL was in [**?** ], where they developed an algorithm that efficiently finds the optimal solution of an MDP of non-stationary domains by using ASP to find the possible trajectories of an MDP. This approach focused more on efficient update of the Q function rather than inductive learning. In order to find stationary sets, an extension of ASP called BC$^{+}$, an action language, was used. BC$^{+}$ can directly translate the agent's actions into ASP form, and provide sequences of actions in answer sets.

# Chapter 6

# Conclusion and Further Research

## 6.1 Contribution

To my knowledge, this is the first time that both symbolic learning method is incorporated into a reinforcement learning to facilitate learning process

## 6.2 Further Research

More complicated environment  
Dynamic environment  
Like moving enermy etc.  
Non-stationality possible to be handled??  
our approach is similar to experience replay ??

### 6.2.1 Value iteration approach

The proposed architecture is not finalised and will be reviewed regularly as we do more research. More research needs to be devoted to finalising the overall architecture, and the following issues in particular need to be considered.

### 6.2.2 Weak Constraint

- Further investigation of whether ILASP can learn the concept of adjacent, which is crucial concept to know in any environment.

- How to generalise the agent's model when the environment changes. The new environment could be very similar to the previous one, or could be a completely different environment thus the agent should create a new internal model rather than generalising the existing model.

- The current proposed architecture is based on Dyna with simulated experiences. However, this might not be the best overall architecture, and the feasibility of using simulated experience with the learnt model with ILASP needs to be further investigated.

- Possibility of using other representational concepts such as *Predictive Representations of State* or *Affordance* [**?** ] for the agent's learning task. These concept have not been considered at the moment, but could help better transfer learning.

- Preparation for a backup plan in case ILASP approach does not work, so that the researchs feasible within 3 months of the researcheriod.

### 6.2.3 probabilistic inductive logic programming

instead of ASP

### 6.2.4 generalisation of the current approach

Learning the concept of being adjacent

## 6.3 Conclusion

# Appendices

## .1 Ethics

To our best knowledge, there is no particular ethical considerations for this particular research listed in Table XX. However, the field of RL ia an active research area and has been increasingly applied in industries these days, and therefore ethical frameworks for RL will be required for both academic research as well as industry applications.
Also the experiemnts of our algorithm were conducted using a game environment rather than real applications (e.g robots).
Rather compared to existing reinforcement learning methodologies.
Also there are a number of AI researchers discussing the ethics of AI in general. Since RL is considered to be part of AI research, these ethical considerations might be also applied.

| | **Yes** | **No** |
|---|---|---|
| **Section 1: HUMAN EMBRYOS/FOETUSES** | | |
| Does your project involve Human Embryonic Stem Cells? | | ✓ |
| Does your project involve the use of human embryos? | | ✓ |
| Does your project involve the use of human foetal tissues / cells? | | ✓ |
| **Section 2: HUMANS** | | |
| Does your project involve human participants? | | ✓ |
| **Section 3: HUMAN CELLS / TISSUES** | | |
| Does your project involve human cells or tissues? (Other than from Human Embryos/Foetuses i.e. Section 1)? | | ✓ |
| **Section 4: PROTECTION OF PERSONAL DATA** | | |
| Does your project involve personal data collection and/or processing? | | ✓ |
| Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)? | | ✓ |
| Does it involve processing of genetic information? | | ✓ |
| Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc. | | ✓ |
| Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets? | | ✓ |
| **Section 5: ANIMALS** | | |
| Does your project involve animals? | | ✓ |
| **Section 6: DEVELOPING COUNTRIES** | | |
| Does your project involve developing countries? | | ✓ |

| | | |
|---|---|---|
| If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned? | | ✓ |
| Could the situation in the country put the individuals taking part in the project at risk? | | ✓ |
| **Section 7: ENVIRONMENTAL PROTECTION AND SAFETY** | | |
| Does your project involve the use of elements that may cause harm to the environment, animals or plants? | | ✓ |
| Does your project deal with endangered fauna and/or flora /protected areas? | | ✓ |
| Does your project involve the use of elements that may cause harm to humans, including project staff? | | ✓ |
| Does your project involve other harmful materials or equipment, e.g. high-powered laser systems? | | ✓ |
| **Section 8: DUAL USE** | | |
| Does your project have the potential for military applications? | | ✓ |
| Does your project have an exclusive civilian application focus? | | ✓ |
| Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items? | | ✓ |
| Does your project affect current standards in military ethics e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons? | | ✓ |
| **Section 9: MISUSE** | | |
| Does your project have the potential for malevolent/criminal/terrorist abuse? | | ✓ |
| Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery? | | ✓ |
| Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied? | | ✓ |
| Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project? | | ✓ |
| **Section 10: LEGAL ISSUES** | | |
| Will your project use or produce software for which there are copyright licensing implications? | | ✓ |

| | | |
|---|---|---|
| Will your project use or produce goods or information for which there are data protection, or other legal implications? | | ✓ |
| **Section 11: OTHER ETHICS ISSUES** | | |
| Are there any other ethics issues that should be taken into consideration? | | ✓ |

**Table 1:** Ethics Checklist

## .2 Learning tasks

This is the full learning task for ILASP in the experiment 1.

## .3 Abduction tasks

This is the full learning task for ILASP in the experiment 1. The syntax and time are added for planning purpose.