Imperial College London

INDIVIDUAL PROJECT REPORT

DEPARTMENT OF COMPUTING

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

Causal Discovery in Explainable Reinforcement Learning

Author:
Siran Shen

Supervisor:
Francesco Belardinelli

June 20, 2022

Submitted in partial fulfillment of the requirements for the BEng degree of Imperial College London

Abstract

Explainability in Reinforcement Learning (RL) is an area drawn many attentions that is fast growing recent years. Humans need to understand how RL systems make decisions to trust and make use of them fully. A lot of approaches have been studied on how to generate interpretation of RL models. Applying causality to construct explanation is particularly intriguing as it builds explanations by mimicking the way humans view the world. The aim of this project is to complete a framework in which given a action influence model of the RL agent, explanations of "why" and "why not" questions in the RL environment can be generated. The action influence model needed in the algorithm requires a directed acyclic graph (DAG) along with its action matrix that represents the casual relation between variables in the environment. Currently, the causal DAG with its associated action matrix can only be generated manually. So to complete the framework, the agent should be able to infer the DAGs and action matrix of the environment on its own. By studying existing casual discovery methods, I propose an algorithm based on the GES algorithm with BIC score. While the outcome produced is rather promising, there are still further improvements and evaluations that can be done. In this case, future work on learning causal models by meta-RL with interventions and computational evaluation algorithm is also discussed.

Acknowledgments

I would like to thank my supervisor, Francesco Belardinelli, for the time and effort spent on helping me with this project, giving me guidance and helping me out when I encounter difficulties during my study.

I would like to thank my parents for all the care and support they give to me.

Finally I would like to thank my boyfriend for all the advice he gave during the project, being patient to me all the time and being by my side to comfort me whenever I need him. I cannot finish this project without his support.

Contents

Chapter 1

Introduction

Reinforcement learning (RL) is one of the most widely used Artificial Intelligence (AI) technologies. In reinforcement learning, RL agent can learn behaviors by observing from the surrounding environment and interact with it based on reward or punishment mechanism [1].

Reinforcement learning has a large number of applications in society, including gaming, healthcare, transports, economics etc [2]. and it interacts strongly with psychology and neuroscience, bringing substantial benefits going both ways. However, to trust RL models, humans need to understand how they make decisions. If an AI system cannot be fully interpreted, it becomes less useful as it is harder to judge if the decisions made are proper or not. Also, under the General Data Protection Regulation (GDPR), calls for transparency for automated decision-making systems rise, which is a fundamental principle for data processing [3]. All in all, it is crucial for AI agents to become explainable so that its potential can be fully discovered [4].

1.1 Problem Statement and Goal

There have always been a lot of approaches studied on how to generate interpretation of RL algorithms. This project aims on explaining RL models with causality. Research in cognitive science proposes that humans understand and represent the world through causes and effects [5] [6]. In the process of understanding the world, causal models are built in humans' minds to encode the causality of events, and these models are used to explain why new events happen and by reference to counterfactuals [7], why those things didn't happen. By mimicking the way humans view the world, in which the world is treated as models where variables interact with each other according to causal relationships, causal models can also be built to generate explanations for RL agents.

To generate explainable RL agents by causality, current research [7] introduces action influence models. Given actions that an agent takes, an action influence model can be defined as a set of structural equations which represent the causal relations between variables in the RL environment. With action influence models, explanations for 'why?' and 'why not?' questions can be generated [7]. Although recent years interest in how to infer action influence models during RL agent training time is increasing, research on it is still significantly less studied. This project aims to fill this gap.

During the process of generating an action influence model, it is essential to provide a directed acyclic graph (DAG), where each edge in the DAG relates to an action taken by the agent and specifies the directed causal relation between variables. So the goal of this project is to generate a DAG with its corresponding action matrix by observations from the RL environment.

1.2 Proposed Solution/Methodology

The main idea to solve this problem is to develop a new algorithm from the existing causal discovery algorithm - Greedy Equivalence Search(GES) Algorithm. Each edge in the final causal DAG will be generated based on the dataset selected by the action taken by the trained RL agent which has the best performance in the environment. The proposed algorithm can be spilt into two parts: CPDAG and Action Matrix Generator and Causal DAG Maker. Further details about this algorithm will be discussed in Section 3.

Three RL environments where the input datasets are generated will be discussed and different RL algorithms will be applied to train agents in the environments. Further details about the RL environments will be discussed in chapter 4.

1.3 Contribution

- Based on current causal discovery method, a new algorithm is designed to successfully generate the causal DAG and the corresponding action matrix.
- The generated DAG and action matrix are applied to generate the structural causal model that is used in explanation generation in [7] and meaningful explanation of "why" and "why not" questions are generated successfully in three chosen environments. The reliability of the developed algorithm is also evaluated by logically analysing the explanations produced.
- Future work on how to improving accuracy of the method designed and computational approach on evaluating the proposed algorithm is also studied in details.

Chapter 2

Background

In this chapter, I will explain the background knowledge needed to understand the method I proposed afterwards. The chapter is arranged as follows: Section 2.1 introduces the basic concept of reinforcement learning, including the algorithms applied to train the agents in the RL environments we chose. Section 2.2 introduces the basic knowledge of causal models. It is important for us to understand this section as it states the key concept of the causal DAG and the action matrix we are going to generate. Section 2.3 compares several causal discovery methods that could be possibly applied in our algorithm. Finally in section 2.4, work done related to this project is studied.

2.1 Reinforcement Learning

This section is mainly based on the book [1].

A reinforcement learning process is a process where one or more agents try to come out with an optimal policy from environment that maximizes the total reward, which is some special value achieved by agents through a sequence of actions they choose.

Definition 1. *A reinforcement agent (RL agent) is the leaner or decision maker that tries to find some algorithm to solve a particular task.*

Definition 2. *A reinforcement learning environment (RL environment) is the thing the RL agent interacts with. Everything surrounding around the agent is called the environment.*

Reinforcement Policy A policy π in reinforcement learning algorithm can be described as a function, mapping from each state(i.e. the environment the agent is currently in) to the probabilities of the agent performing each possible action under

this state. At time t, given policy π the agent follows, $\pi(a \mid s)$ is defined as the probability when the action currently taken by the agent A_t is some action a, given the current agent state S_t is some state s.

A RL problem can be described as a Markov Decision Process (MDP) mathematically. In MDP, agent interacts with the environment continually. The interactions are illustrated in Figure 2.1. The agent performs an action according to the state, and the environment responds to its action and pass a new state to let agent observe. Each action also changes the total reward, which is aimed to be maximized by the agent.

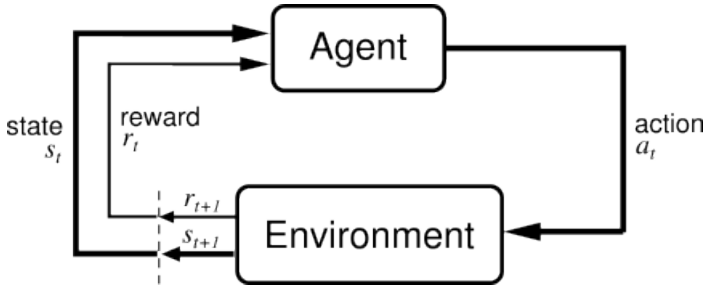

Figure 2.1: Agent-Environment Interactions in MDP

Definition 3. *A Markov Decision Process (MDP) is a 5-tuple $M = \langle S, A, R, P, \gamma \rangle$, where*

- *S, A, and R separately represent the sets of states, actions, and rewards. The number of elements in all sets is finite;*
- *P is the transition probability:*

$$p(s', r \mid s, a) = Pr(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a),$$

for all $s', s \in S, r \in R, a \in A(s)$. It presents the probability of some particular state s' and reward r values at time t given the previous state s and the action taken a; γ is the discount rate ($\gamma \in [0, 1]$), it decides the present value of future rewards.

Given a sequence of rewards after time step t: $R_{t+1}, R_{t+2}, R_{t+3}, ...$ and a discount rate γ, the maximized discounted return over future, which is a function based on this reward sequence, can be received if the agent selects correct actions, where

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

For MDPs, the value function $v_\pi(s)$ defines the expected return starting from state s under policy π.

Given policy π and state s, the following relationship holds between s and all of its successor states:

$$\begin{aligned} v_\pi(s) &= E_\pi \left[G_t \mid S_t = s \right] \\ &= E_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s \right] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma E_\pi \left[G_{t+1} \mid S_{t+1} = s' \right] \right] \\ &= \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right], \end{aligned} \tag{2.1}$$

where $E_\pi \left[. \right]$ expresses the expected return value under policy π and time step t. This expression indicates how good it will be in state s when policy π is followed.

Equation 2.1 is the Bellman equation for v_π, which describes the relationship between one state's value and its successor states' values, weighting each possible successor state by the number of times it could possibly occur. It states the value of start state s must equal the sum of the value of next state and the following expected reward. Figure 2.2 is the backup diagram of state value v_π, which presents the backup operations transferring the value of one state's successor states back to it.

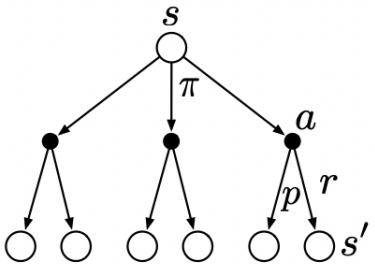

Figure 2.2: Backup Diagram for v_π by [8] . A white circle is a state and a black circle is state–action pair: (s, a). Beginning from the top node(state s), according to policy π, the RL agent can take any from some action sets (three shown in the diagram). The environment will respond to each action with one from several next states, s' (two in the diagram), with a reward, r, based on the next state's dynamics which is given by the function p.

Same as v_π, the action function $q_\pi(s, a)$ defines the expected return by taking action a, staring from state s under policy π:

$$q_\pi(s, a) = E_\pi \left[G_t \mid S_t = s, A_t = a \right] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right],$$

It indicates how good it will be with action a in state s under policy π.

2.1.1 Bellman Optimality Equation

In this section we are going to introduction the Bellman optimality equation, which is used to get optimal policy in different RL algorithms applied in our chosen RL environments. Further details of the RL algorithms will be introduced in section 2.1.2.

As mentioned before, the task of reinforcement learning algorithm is finding an optimal policy, defined as the policy is equal to or better than all of the other policies. More than one optimal policy can be found and we denote all of them by $\pi_\star$. Different optimal policies have the same state-value and action-value function. They are called optimal state-value and action-value function separately.

An optimal state-value function is:

$$v_\star = \max_\pi v_\pi(s),$$

for all $s \in S$.

An optimal action-value function is:

$$q_\star = \max_\pi q_\pi(s, a),$$

for all $s \in S$ and $a \in A(s)$.

There are Bellman optimality equations for both $v_\star$ and $q_\star$. An Bellman optimality equation intuitively states that under an optimal policy, the value of a state s must equal the expected return of the best action from s.

The Bellman optimality equation for $v_\star$ is

$$\begin{aligned} v_\star &= \max_{a \in A(s)} q_{\pi\star}(s, a) \\ &= \max a E_{\pi\star} \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a \right] \\ &= \max a E \left[R_{t+1} + \gamma v_\star S_{t+1} \mid S_t = s, A_t = a \right] \\ &= \max a \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma v_\star(s') \right] \end{aligned}$$

The Bellman optimality equation for $q_\star$ is

$$\begin{aligned} q_\star(s, a) &= E \left[R_{t+1} + \gamma \max a' q_\star(S_{t+1}, a') \mid S_t = s, A_t = a \right] \\ &= \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma \max a' q_\star(s', a') \right] \end{aligned}$$

In Figure 2.3, the relationships between the value of one state (or state–action pair) and the value of its successor states (or state–action pairs) in the Bellman optimality

equations are graphically presented. These diagrams are the same as the $v_\star$ backup diagram presented earlier apart from arcs appears when the RL agent is making choice. So at each step, instead of taking the expected value given some policy, the maximum over that choice will be taken.

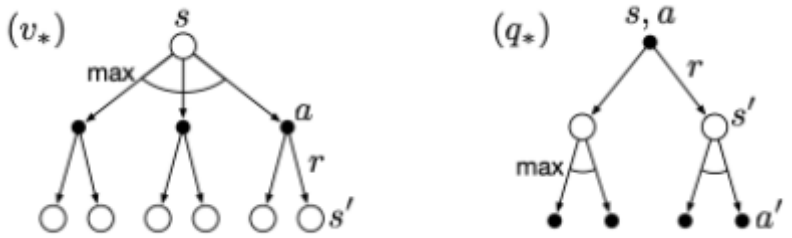

Figure 2.3: Backup Diagrams for $v_\star$ and $q_\star$ by [8]

2.1.2 Reinforcement Learning Algorithms

In this section, I will introduce algorithms that train the agents in the three different RL environments in which my algorithm are applied to. Evaluation benchmark is also set up with the environments using these algorithms.

Q-Learning

Q-learning algorithm is the RL algorithm used in the environment - Taxi Problem. Further details about this environment can be found in section 4.1.

Q-learning algorithm is a Q-value RL algorithm [1]. At each state, each action is assigned with a Q-value which intuitively indicates how good the action is at a specific state. At each step, the action having the maximum Q-value is always chosen, then at the next state, the Q-value of this state-action pair is updated with this maximum Q-value.

The key idea of Q-learning is that we can recursively calculate $Q_\star$ using the Bellman equation: $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$. Algorithm 1 illustrates formally how the Q-learning algorithm works [9].

Deep Q-Network (DQN)/ Deep Q-learning

DQN(Deep Q-learning) algorithm is the RL algorithm used in the environment - LunarLander. Further details about this environment can be found in section 4.3.

Algorithm 1 Q-Learning

Require: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Ensure: Off-policy control for estimating $\pi \simeq \pi_\star$
Initialize $Q(s, a)$: $\forall s \in S^+$, $a \in A(s)$ arbitrarily.
Set $Q(terminal, \cdot) = 0$
for each episode **do**
    Initialize S;
    for step = 0 to T **do**
        Choose A from S using policy derived from Q (e.g. $\epsilon - greedy$);
        Take action A, observe R, S';
        $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$;
        $S \leftarrow S'$;
    end for
end for

DQN algorithm is about applying deep neural networks to RL environment. Figure 2.4 demonstrates the process in detail.

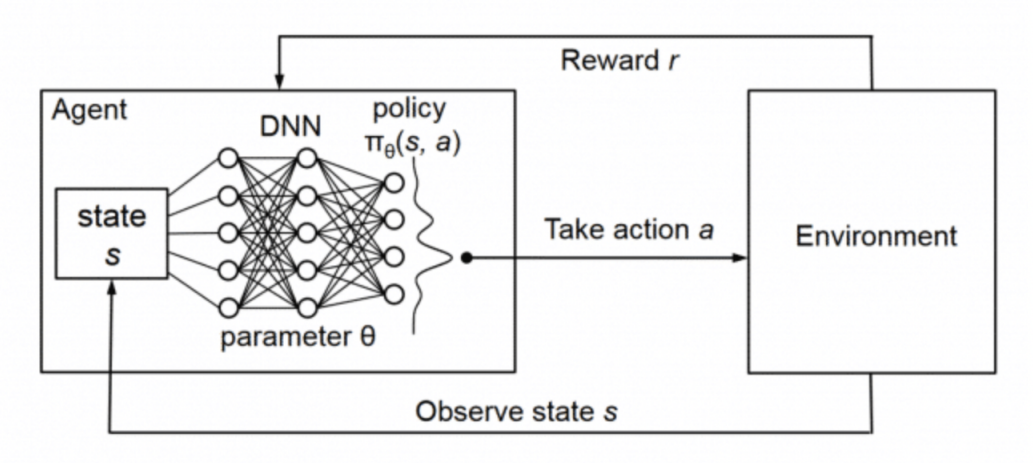

Figure 2.4: interaction between agent and environment in DQN algorithm

In DQN, a deep neural network, which can be thought as a black-box is used to estimate the Q-value function. Taken the environment state, Q-value for every possible action is returned as output. Then like in Q-Learning algorithm, the action having the greatest Q-value will be chosen.

The process of DQN algorithm can be described as following [10]:

1. All the previous experience is stored.

2. The next action is decided based on the highest output value of the network.

3. From Bellman equation, the Q-value update equation is derived: $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$. Current state is updated to the next state by applying the action that has the best Q-value.

4. The network then updates its gradient by backpropagation and will finally converge.

Figure 2.5 compares Q-learning and DQN algorithm.

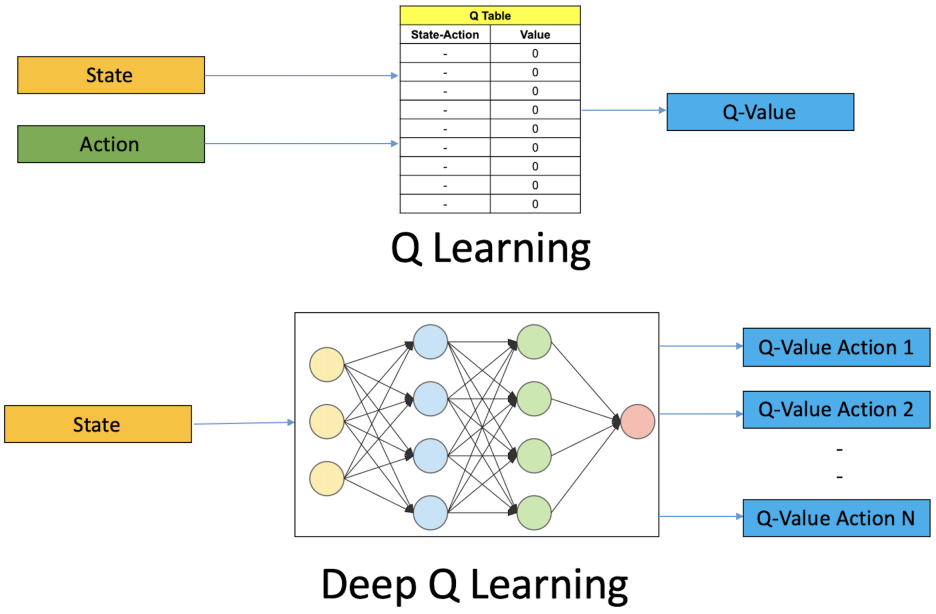

Figure 2.5: comparison between Q-learning and DQN(Deep Q-learning) algorithm

REINFORCE algorithm

REINFORCE algorithm is the RL algorithm used in the environment - CartPole. Further details about this environment can be found in section 4.2.

In RL algorithm, REINFORCE algorithm belongs to the category of policy gradient algorithms [1]. Policy gradients are different from Q-value algorithms because instead of approximating Q-values of actions at each state, policy gradients will learn a parameterized policy. So the policy output is a probability distribution over actions rather than a Q-value table over action-state pair.

The steps of REINFORCE can be described as following [11]:

1. Initialize a Random Policy, which is a neural network(NN) taking a state as input and returning the probability of actions;

2. The RL agent uses the policy to play N steps in the environment. Action, action probabilities and reward will be recorded;
3. For each step, discounted reward for each step is calculated by backpropagation;
4. Expected reward G is calculated;
5. Policy weights are adjusted by backpropagate error in NN to increase G;
6. Repeat from step2.

2.2 Causal Models

A causal model is a model which can represent the casual relationship between variables in one scenario. There are two kings of variables, endogenous and exogenous. Endogenous are those whose values are decided by factors inside the model, whereas exogenous are those whose values are decided by factors outside the model [12]. Causal model helps to infer causality from statistics and is a good way to generate explanation from RL environment as it describes the environment in a similar way as humans understand the world [5] [13].

2.2.1 Structural Causal Model(SCM)

One common way to represent a casual model is to use a structural equation set. SCM is necessary in producing action influence model which is used to generate explanation in RL environment [7].

Definition 4. *A structural causal model (SCM) is a tuple $M = (S, F)$, where*

- *S is a signature, represented by a tuple (U, V, R), in which U and V are the set of exogenous and endogenous variables separately, R denotes the values of every variable in set $U \cup V$;*
- *F is the structural equation set calculating the value of each endogenous variable given some other variables in the model [14].*

2.2.2 Graphical Causal Models

A casual model in graphical form is a DAG in which each node is a random variable and each edge represents a causal relationship. If in the structural causal model, a

variable X is one of the arguments in F to calculate variable Y, then an edge from variable X to variable Y will be shown in the graphical causal model [15].

Definition 5. *An graph G is a pair (V, E), where*

- *V is the node set in G;*
- *$E \subseteq V \times V$ is the edge set in G;*
- *The pair $(u, v) \in E$ indicates the directed edge: $u \to v$, whereas set $\{(u, v), (v, u)\} \in E$ indicates the undirected edge: $u - v$;*

Definition 6. *A Undirected graph (UG) is a graph only having directed edges.*

Definition 7. *A Directed Acyclic Graph (DAG) is a graph without cycles and only having directed edges.*

Definition 8. *A causal matrix is a matrix representation of its casual model graph.*

In this project, a casual model graph with N variables is represented by a $N \times N$ matrix where $Mat[i, j] \neq 0$ implies edge $i \to j$ and $Mat[i, j] \neq 0 \ \ \& \ \ Mat[j, i] \neq 0$ implies edge $i - j$. An empty graph is a matrix of zeros. A working example is the casual matrix presented in Figure 2.7.

Definition 9. *An action matrix is in the same form of its associated causal matrix, where*

- *each non-zero element in the matrix represents the index of the action performed on that edge.*
- *A working example is the action matrix presented in Figure 2.7.*

2.2.3 Action Influence Model

An action influence model for a RL agent is one SCM with an associated set of actions. It is necessary in generating explanations for a RL agent [7].

Definition 10. *An action influence model is a tuple (S_a, F), where*

- *S_a is a signature, represented by a tuple (U, V, R, A);*
- *A represents the set of actions while U, V, R are the same as in a SCM;*
- *F is a structural equation set in which there is a set of structural equations for each variables $X_r \subseteq V$. Each set of the structural equations implies a unique action set which influences X.*

- *For an action $a \in A$, a function $F_{X.A}$ defines the causal effect on X under a.*

Figure 2.6 is a graphical representation of one action influence model in game StarCraft. In the figure, each edge implies there is a causal effect from one variable to another, associated with an unique action.

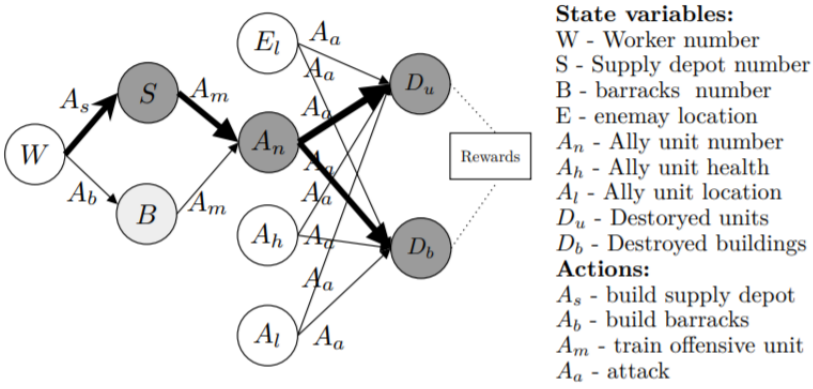

Figure 2.6: StarCraft Action Influence Graph by [16]

Figure 2.7 is the associative causal model graph and action matrix. The graphical causal model is represented in matrix form on the left and the corresponding action matrix is represented on the right.

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 91 & 42 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 477 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 477 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13 & 13 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13 & 13 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13 & 13 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13 & 13 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2.7: Causal Matrix and Action Matrix of StarCraft Action Influence Graph by [16]

2.2.4 Markov Equivalence Class(MEC) and Completed Partially Directed Graph (CPDAG)

The following notions are essential in GES algorithm, which our algorithm is derived from. Further details about GES algorithm will be introduced in section 2.4.1.

Sometimes, different causal models can imply the same conditional independencies. This causal model set is called Markov Equivalence Class.

Notions

- A skeleton is a underlying undirected graph of a causal model.
- A v-structure between node u, w, z is in form of $u \rightarrow w \leftarrow z$ [15].

According to theorem(Verma Pearl, 1991), Two graphical casual models $G1$ and $G2$ are Markov equivalent if and only if they have the same skeleton and the same v-structures [17].

An example is shown in Figure 2.8. As we can see, D_1, D_2 and D_3 share the same skeleton and v-structures ($1 \rightarrow 4 \leftarrow 2$, $3 \rightarrow 4 \leftarrow 2$, $1 \rightarrow 4 \leftarrow 3$).

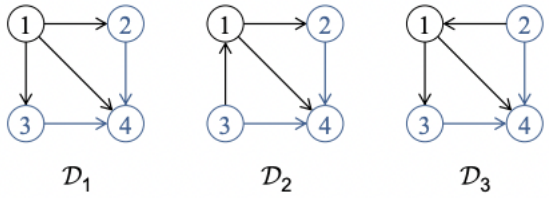

Figure 2.8: three Markov equivalent DAGs – explain why these three are equivalent

Completed Partially Directed Graph (CPDAG) is used to represent the union of a Markov equivalence class [18]. It combines both undirected and directed edges. In one CPDAG, every v-structure in its corresponding Markov equivalence class is kept and all other directed edges are changed into undirected edges.

An example is shown in Figure 2.9. G is the CPDAG of D_1, D_2 and D_3. The v-structures ($1 \rightarrow 4 \leftarrow 2$, $3 \rightarrow 4 \leftarrow 2$, $1 \rightarrow 4 \leftarrow 3$) are kept in G and directed edges among node 1, 2, 3 are changed into undirected ones.

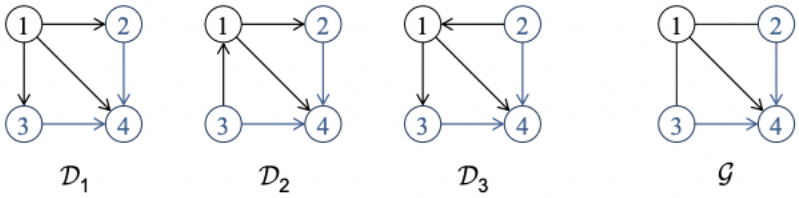

Figure 2.9: G is CPDAG of $D_{1,2,3}$

Modeling Interventions

Interventions can be represented in graphic casual models. A interventions is modelled by adding a node and an edge to the model, indicating the intervention variable as well as the variable it acts on [19].

There are two types of interventions: hard interventions and soft interventions. A hard intervention breaks all other edges in the model to the child node of the intervention variable while a soft intervention add an influence on the child node without breaking the effect from other causes.

There are three causal structures in graphic casual model which are shown in Figure 2.10. The v-structure mentioned in previous section is also called confounder. By applying hard interventions, we can break the influence of confounders.

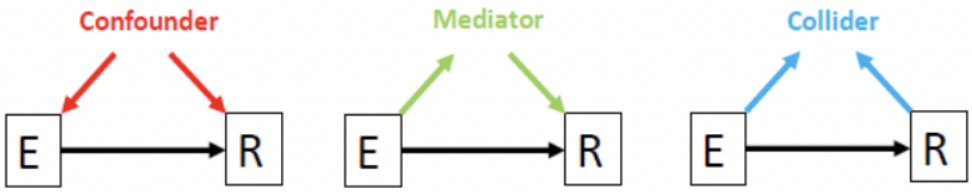

Figure 2.10: three casual structures

For example, 'smoking' is a confounder of 'Yellow Teeth' and 'Lung Cancer'. By adding a hard intervention of variable 'Paint' to 'Yellow Teeth', the edge from 'Smoking' to 'Yellow Teeth' is broken, so no conditional independence will be obversed between 'Yellow Teeth' and 'Lung Cancer'.

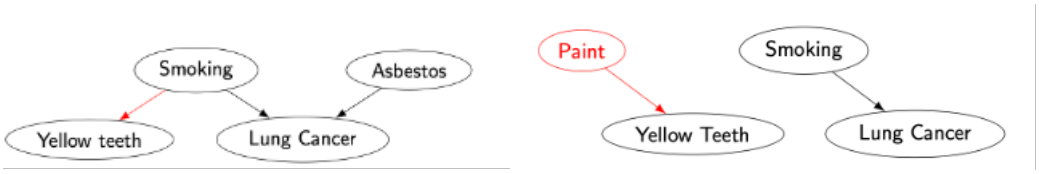

Figure 2.11: an example before and after an hard intervention

2.3 Causal Discovery

Causal discovery usually means the task to find the underlying causal model between variables in a system. It is a fundamental task in all kinds of disciplines in science area.

Traditionally, causal relations discovery contains the use interventions (briefly mentioned in section 2.2.4), or randomized experiments, which is too expensive, or even impossible under many circumstances. Therefore, discovering causal relations by analyzing purely observational data has drawn much attention [20].

There are two types of causal discovery methods [19]: method with interventions and method without interventions. Causal discovery methods without interventions can be further split into three sub-groups: constraint-based method, score-based method and hybrid method.

In this project, my method to generate the graphical causal model and associative action matrix is based on GES algorithm, which is a score-based method without interventions.

2.3.1 Constraint-Based vs Score-Based Methods without Interventions

In this section, I will introduce and compare constraint-based methods and score-based methods. The advantages and disadvantages of both of them will be discussed briefly.

Constraint-based Methods This type of methods is based on the fact that the independence between variables achieved by statistical conditional independence(CI) tests can be directly shown on the causal diagram [21].

Constraint-based methods usually consist of two steps:

1. Through the CI test, the skeleton of the causal structure is learned, that is, the undirected graph that does not indicate the causal direction.
2. Then the causal direction of edges is indicated, usually based on the collision structure(collider), because among the three causal structures(shown in Figure 2.10), only the direction of the collision structure is different from the other two. Figure 2.12 demonstrates in details the relationship between causal structures and Observed CIs.

True DAG	$A \rightarrow B \rightarrow C$	$A \rightarrow B \leftarrow C$
Observed CIs	$A \perp\!\!\!\perp C \mid B$	$A \perp\!\!\!\perp C \mid \emptyset$
Set of DAGs in MEC	$A \rightarrow B \rightarrow C$ $A \leftarrow B \leftarrow C$ $A \leftarrow B \rightarrow C$	$A \rightarrow B \leftarrow C$
CPDAG	$A - B - C$	$A \rightarrow B \leftarrow C$

Figure 2.12: the relationship between causal structures and Observed CIs

Score-based Methods Each graph in this type of algorithm will obtain a score and the graph with highest score is chosen after going through all the possible graphs. A optimal score function is needed to make this algorithm work [19].

The advantage of this type of method is that it uses goodness of fit, which has less restriction, instead of conditional independent in constraint-based algorithm, but the disadvantage is that MEC is still indistinguishable. Another disadvantage is this kind of algorithms needs to search the entire graph space to find the optimal score. It has extremely high complexity and is easy to fall into local optimum as it is a NP-hard and NP-complete problem.

2.4 Related Work

This part is discussed mainly from related papers. I am going to discuss related work done in reinforcement learning area first, then discuss current causal discovery methods and preliminaries in generating explanations for RL agents.

2.4.1 Greedy Equivalence Search(GES) Algorithm

This section introduces the GES algorithm. As mentioned before, GES algorithm is the score-based causal discovery method that my new algorithm is based on.

The GES algorithm has three stages: forward, backward and turning, performing add edges remove edge and turn edge operations separately. The input of this algorithm is a dataset D with N variables in it along with an empty graph G that will have the same variables as in D, and output of this algorithm is a CPDAG C with a score assigned to it which indicates how good the CPDAG is at representing the actual casual graphic model.

The GES algorithm is implemented in the following way [22]:

- It starts with the forward stage, performing a round of add edge operations to all the edge candidates between variables that have not had an edge between them. At each step, it calculate if adding the edge will increase or decrease the score of the graph G. If the score is increased by one adding edge operation, then adding this edge is marked as a valid operation. After going through all the available edges in graph G, the valid operation with the highest score will be added to G. The graph G after the edge is added is a PDAG, so after adding the edge, a algorithm will be applied to transfer the PDAG to a CPDAG as the GES algorithm cannot distinguish Markov equivalence classes(discussed in section 2.3.1). This step will repeat until the total score of G cannot be increased any more.
- Then the backward stage starts. The same approach will be implemented as in forward stage, but backwards instead. At each step, it calculates if deleting an existing edge in G will increase the score.

- Finally, the turning stage runs to see if changing the direction of a directed edge in G will increase the score of the graph. This stage is not part of the original algorithm, but is introduced later in [23].

Definition 11. *A likelihood function $L = p(x \mid \theta, M)$, where*

- *x is the data obtained by observation;*
- *θ the current values of the parameter set in M;*
- *A likelihood function represents the probability of obtaining the observed data x given the current value of parameters θ in model M.*

Currently, there are multiple score functions that can be applied in this algorithm. In this project, the Bayesian Information Criterion(BIC) is used. The BIC is defined as below:

$$BIC(G) = klog(n) - 2log(\hat{L}(\hat{\theta}))$$

In BIC score function [24] , k is the number of parameters estimated by the model M and n represents the sample size. $\hat{L}(\hat{\theta})$ is the highest value of the likelihood function of M, where $\hat{\theta}$ is the values of parameters in M maximizing the likelihood function L.

Why choose GES algorithm?

GES algorirhm is one of the most famous score-based causal discovery algorithm. In the second stage of the Causal DAG Maker, which is the second part of our algorithm designed to generate the causal DAG along with its action matrix, the final edge of the DAG is decided by the score of each existing edge, so this score-based algorithm is chosen and used in CPDAG and action matrix generator, which is the first part of our algorithm.

2.4.2 Learning Explanations from Casual Graphic Model

This part is based on paper [7].

In this section, I will introduce the process of how to generate ”why” and ”why not” explanations after the causal DAG and action matrix is given.

Given a causal graphic DAG specifying causal direction between variables along with its action matrix, during the training phase of a RL agent, structural equations of

this agent can be learned as a set of regression models with multiple variants. Then explanations of "why A" and "why not A" questions (A is some action in the RL environment) can be produced given the SCM generated above.

The process of explanation generation can be treated as a "black box" as it is not the main focus of this project. We will only use the explanation generated to analyse if the causal graphic model and the action matrix we get by the designed method are accurate enough so that meaningful explanation can be produced.

Chapter 3

Methodology

In this chapter, I will introduce the algorithm designed to automatically generate the casual DAG along with the action matrix of an RL environment.

The algorithm can be divided into two parts. The first part is to generate a CPDAG with its action matrix based on GES algorithm, which is introduced in section 3.1. The second part is to make the causal DAG from the CPDAG and action matrix generated in the first part, which is introduced in section 3.2.

3.1 CPDAG and Action Matrix Generator

In this section, I will introduce the algorithm used to generate a CPDAG along with the action matrix given the observed data obtained from runnning the trained agent in the RL environment.

Algorithm 2 presents in details how the algorithm actually works.

The algorithm is based on GES algorithm (details of GES algorithm are shown in section 2.4.1). However, instead of running the algorithm directly on the input dataset D, the input data is first grouped by actions (line 4). Then at each step where add edge, remove edge or turn edge operations is performed, rather than calculating the graph score based on the whole input dataset, for each edge, all the datasets grouped by different actions will be gone through one by one, used as the input data to calculate the graph score (from line 12 to line 18). The best score among all the scores gotten from datasets will be marked as the final score of applying the operation to that edge (from line 19 to line 23). The edge will then be operated in the graph, and the action associated with the dataset will be seen as the action along with that casual relationship and be operated(added/deleted) in the action matrix (from line 25 to line 28).

Algorithm 2 CPDAG and Action Matrix Generator

Require: Dataset D
Ensure: A CPDAG G representing the causal relationship in D; Action matrix M

Initialize an empty graph G with D;
Initialize the score of G: $L \leftarrow 0$; ▷ L is the score of G
Initialize an empty matrix M with D;
Get a set of datasets S by grouping data in D by actions; ▷ each dataset in S is assigned with an unique action
for each stage in forward, backward, turning **do**
    while the number of edge candidates in $G \neq 0$ **do**
        Get edge candidate set Es;
        for each edge E in Es **do**
            $W \leftarrow 0$; ▷ W is the highest score of G that can be gotten by Es
            $Z \leftarrow 0$; ▷ Z is the highest score of G can be gotten by operating on edge E
            Initialize action A along with E;
            for each dataset in S **do**
                Calculate the score C based on this dataset;
                if $C > Z$ **then**
                    $Z \leftarrow C$;
                    Assign the action associated with this dataset to A;
                end if
            end for
            if $Z > W$ **then**
                $W \leftarrow Z$;
                Store E as $E^\star$;
                Store A as $A^\star$;
            end if
        end for
        if $W > L$ **then**
            $L \leftarrow W$;
            Do operation to $E^\star$;
            Add or delete $A^\star$ in M; ▷ The actual operation depends on which stage it is in
        else
            End While loop;
        end if
    end while
end for

3.2 Casual DAG Maker

The main idea to make the causal DAG is by running the CPDAG and action matrix generator M times(for instance 100 times), the DAG can be generated from the M CPDAGs. Assume each CPDAG has N variables, starting with an empty graph G with N variables, the algorithm has two stages:

- The first stage is to add the M CPDAGs to G one by one. After this stage, between two varibales in G, there will be multiple edges, each one associated with one unique set of action, direction and score. Algorithm 3 shows in details how this stage works.

Algorithm 3 Adding CPDAGs to into graph G

Require: a set of CPDAGs-Score pairs S; ▷ every edge in CPDAG is assigned with an unique action
Ensure: A graph G in which there are multiple edges between two nodes

```
1: Initialize an empty graph G;
2: for each CPDAG in S do
3:     for each edge E in CPDAG do
4:         if E is directed edge between nodes v and u then
5:             ADDEDGE(E, G)
6:         else                    ▷ If E is undirected, treat E as two directed edges
7:             Set E' = v ← u;
8:             ADDEDGE(E', G);
9:             Set E'' = v → u;
10:            ADDEDGE(E', G);
11:        end if
12:    end for
13: end for
14:
15: function ADDEDGE(E, G)
16:    if G contains E with its action then
17:        Add the score of E to that edge in G;
18:    else
19:        Add E with its action and score to G;
20:    end if
21: end function
```

- The second stage is to get the final causal DAG by merging all the edges between two variables into one unique directed edge. Algorithm 4 shows in details how this stage works. The threshold parameter H in the algorithm is used to decide if the edge will be kept in the final DAG produced. If the score of the edge is smaller than H, then it will be removed from the graph (from line 5 to line 6). In the environments chosen, we set $H = 0.1$, which is a pretty

small value as we want to remove all the impossible edges and keep as many edge as possible for further evaluation at the same time.

Algorithm 4 Merging edges between two variables into one

Require: A graph G where there are multiple edges between two variables and every edge is assigned with an unique action and score
Ensure: A DAG $G^\star$ in which every edge is assigned with an unique action and score
1: Set threshold H;
2: Get the maximum score M in G;
3: **for** each edge E in G **do**
4: Divide the score S' of the edge by M;
5: **if** $S' < H$ **then** ▷ Thresholding S'
6: Delete E in G;
7: **end if**
8: **end for**
9: Initialize an empty graph $G^\star$;
10: **for** each pair of nodes u and v in $G^\star$ **do** ▷ Pick up the best edge between two nodes
11: Initialize score between u and v: $S = 0$;
12: **for** each edge E between u and v **do**
13: **if** score of $E > S$ **then**
14: Set E as the edge between u and v in $G^\star$;
15: **end if**
16: **end for**
17: **end for**

Chapter 4

Environments and Implementation

In this chapter, three RL environments are introduced separately: Taxi Problem (in section 4.1), CartPole (in section 4.2) and LunarLander (in section 4.3). In order to make sure our methodology designed can be widely applied to all RL environments with a wide range of variable numbers and different RL algorithms, three different RL environments are chosen. Each environment uses a learning algorithm and observation state different from others. All the three environments along with parameters in them are set by the official Gym OpenAI documentation [25]. In each section, we also present how to implement the methodology introduced in section 3 in details.

4.1 Taxi Problem

The environment of the taxi problem contains a 5×5 grid world. Each square on the grid world is a location. Obstacles are set on the grid and the taxi cannot move across obstacles. There are four special locations, denoted by letter R(red), G(green), Y(yellow), and B(blue). The taxi is considered as the agent. At the start of each episode, the taxi is initialised at a random square and the passenger is located randomly at one of the four special locations. The taxi's first task is driving to the passenger's location and picking up the passenger. Then it drives the passenger to the destination location, which is another one of the four special locations and the episode terminates.

An illustration of the environment in shown in Figure 4.1. The four locations can be seen labelled as R, G, Y, B at coordinates (0,0), (0,4), (4,0) and (4,3) separately in the grid. The destination location is at (0,0) and the passenger is located at (4, 0) at start.

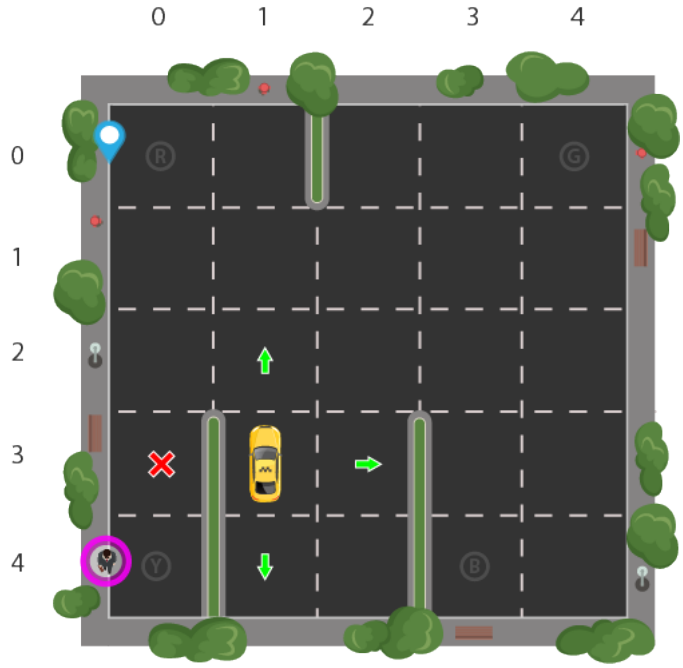

Figure 4.1: An illustration of the initial Environment of Taxi Problem

Actions The taxi has 6 actions in the environment, each action is indexed by an integer: 0. move south, 1. move north, 2. move east, 3. move west, 4. pickup passenger and 5. drop off passenger.

State space The taxi can be at any location in the grid, so it has 5×5 locations in total. There are 4 destination locations, which are the four special locations and the passenger has 5 locations (4 special locations and the location which is inside the taxi).

The environment represents the taxi location with the grid coordinates. The passenger and destination locations are represented by indexes which are shown as below:

- Passenger location: 0. R(red), 1. G(green), 2. Y(yellow), 3. B(blue) and 4. in taxi.
- Destination location: 0. R(red), 1. G(green), 2. Y(yellow), and 3. B(blue).

A State space is the set of all states in the environment. A state is a represented by a tuple: (taxi_row, taxi_col, passenger_location, destination_location)

Rewards At every step, the taxi agent will receive a -1 reward if other reward is not triggered. If the taxi pick up or drop off the passenger successfully, it will receive a +20 reward. When it does ”pickup” and ”drop-off” actions wrongly, it will receive a -10 reward.

4.1.1 Implementation

To solve the taxi problem task, I used the Q-learning algorithm implemented in [26].

First the agent in taxi problem is trained by Q-learning algorithm. Then the game is run for 100 episodes with the trained agent. During each episode, for each step, one action will perform and the state alone with the total reward before and after the action performed will be recorded. We encode (taxi_row, taxi_col) in the state to an integer representing taxi_location.

There are 6 actions in the environment. We refer actions: 0. move south, 1. move north, 2. move east, 3. move west all as one action: move, indexed by 6.

So for each step, two tuples: (taxi_location_before, passenger_location_before, destination_location_before, total_reward_before, action), (taxi_location_after, passenger_location_after, destination_location_after, total_reward_after, action) will be recorded.

After 100 episodes, a data set is gotten with 5 columns: taxi_location, passenger_location, destination_location, total_reward, action for each. Then then dataset is put into the CPDAG and Action Matrix Generator and the final DAG is made.

The final DAG produced will be a 3×3 matrix in form of:

$$\begin{bmatrix} taxi\_position & passenger\_position & destination\_location \\ passenger\_position & \cdots & \\ destination\_location & \cdots & \end{bmatrix}$$

4.2 CartPole Problem

The environment of the CartPole problem contains a pole attached to a cart moving along a track which is frictionless. In each episode, the pole is placed upright on the cart and the joint between them are un-actuated. The task is to keep the pole balanced as long as possible, by applying forces on the cart. The force applied can be in left or right direction force. The episode ends under three circumstances: 1. the angle of the pole is over $\pm 12^{\circ}$, 2. the center of the cart is outside the view window, 3. the total episode number is over 500.

An illustration of the intial environment is in shown in Figure 4.2.

Actions There are two actions in this environment, each is indexed by an integer: 1. push cart to the left, 2. push cart to the right.

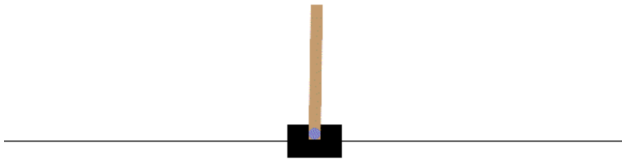

Figure 4.2: An illustration of the initial Environment of the CartPole Problem

State space A state in CartPole environment is a tuple containing four elements, each element is indexed by an integer: 0. the position of the cart, 1. the velocity of the cart, 2. the angle of the pole, 3. the angular velocity of the pole.

All elements are initiated randomly in range (-0.05, 0.05) at the beginning of each episode.

Rewards Each frame is considered as a step. As the task is to keep the pole balanced for as long as possible, every step, including the termination step, receives a $+1$ reward.

4.2.1 Implementation

To solve the CartPole problem task, I used the REINFORCE implemented in [27].

First the agent in CartPole problem is trained by REINFORCE algorithm. Then the game is run for 100 episodes with the trained agent. During each episode, at each step, one action will perform and the state alone with the total reward before and after the action performed will be recorded.

Both the state and actions defined in section 4.2 will be used, so for each step, two tuples: (cart_velocity_before, cart_position_before, pole_angle_before, pole_angular_velocity_before, total_reward_before, action), (cart_velocity_after, cart_position_after, pole_angle_after, pole_angular_velocity_after, total_reward_after, action) will be recorded.

After 100 episodes, a data set is gotten with 6 columns: cart_velocity, cart_position, pole_angle, pole_angular_velocity, total_reward, action for each. Then then dataset is put into the CPDAG and Action Matrix Generator and the final DAG is made.

The final DAG produced will be a 4×4 matrix in form of:

$$\begin{bmatrix} & cart\_velocity & cart\_position & pole\_angle & pole\_angular\_velocity \\ cart\_position & \cdots & & & \\ pole\_angle & \cdots & & & \\ pole\_angular\_velocity & \cdots & & & \end{bmatrix}$$

4.3 LunarLand Problem

The environment of the LunarLander problem contains a viewport and the lunar lander is considered as the agent. At each episode, the lander starts off the top center of the view window and needs to move towards the landing pad and land safely, which means the lander legs touch the moon land. The episode terminates under three circumstances: 1. the lander crashes which means instead of the two legs, the body of the lander hit the moon; 2. the lander gets outside of the view window; 3. the lander is not awake, which means the lander does not start moving.

An illustration of the intial environment is in shown in Figure 4.3 and an illustration of the lander landing successfully is in shown in Figure 4.4.

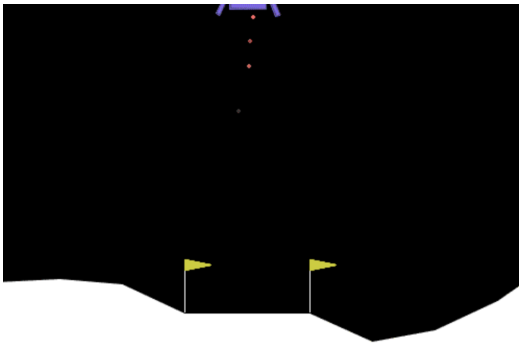

Figure 4.3: An illustration of the initial Environment of LunarLander Problem

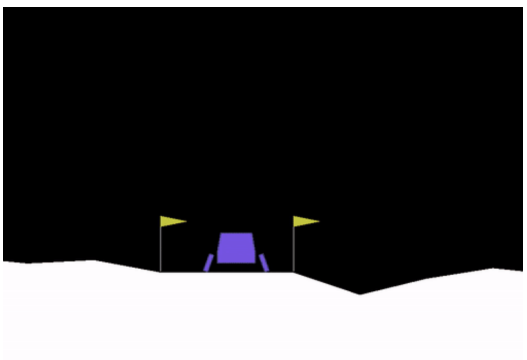

Figure 4.4: An illustration of the lander landing at the landing pad successfully

Actions The lunar lander has 4 actions in the environment, each action is indexed by an integer: 1. do nothing, 2. fire left orientation engine, 3. fire main engine, 4.

fire right orientation engine.

State space A state in LunarLander environment is a tuple containing eight elements, each element is indexed by an integer: 0. the coordinate of the lander in x-axis, 1. the coordinate of the lander in y-axis, 2. the lander's linear velocity in x-axis, 3. the lander's linear velocity in y-axis, 4. the angle of the lander, 5. the angular velocity of the lander, 6. if the left leg of the lander contacts the ground, 7. if the right leg of the lander contacts the ground.

Rewards Each frame is considered as a step. At every step, the lander receives a +100 point reward when moving from the start point to the landing place. If it moves away from the landing place, it loses the reward. If the lander crashes, the reward loses an additional 100 points. If it comes to rest, the reward adds an additional +100 points. +10 points is added if any leg of the lander contacts with the ground. Firing the main engine loses 0.3 points. Firing the side engine loses 0.03 points. the lander lands successfully within the landing place is +200 points.

4.3.1 Implementation

To solve the LunarLand problem task, I used the Deep Q-Network algorithm implemented in [28].

First the agent is trained by DQN algorithm. Then the game is run for 100 episodes with the trained agent. During each episode, for each step, one action will perform and the state alone with the total reward before and after the action performed will be recorded.

Both the state and actions defined in section 4.3 will be used, apart from the two elements: if the left leg of the lander touches the ground, and if the right leg of the lander touches the ground.

So for each step, two tuples: (lander_xCoor(dinate)_before, lander_yCoor_before, lander_xV(elocity)_before, lander_yV_before, lander_angle_before, lander_angularV_before, total_reward_before, action), (lander_xCoor_after, lander_yCoor_after, lander_xV_after, lander_yV_after, lander_angle_after, lander_angularV_after, total_reward_after, action) will be recorded.

After 100 episodes, a data set is gotten with 8 columns: lander_xCoor, lander_yCoor, lander_xV, lander_yV, lander_angle, lander_angularV, total_reward, action for each. Then then dataset is put into the CPDAG and Action Matrix Generator and the final DAG is made.

The final DAG produced will be a 6×6 matrix in form of:

$$\begin{bmatrix} xCoor & yCoor & xVelocity & yVelocity & angle & angularVelocity \\ yCoor & \cdots & & & & \\ xVelocity & \cdots & & & & \\ yVelocity & \cdots & & & & \\ angle & \cdots & & & & \\ angularVelocity & \cdots & & & & \end{bmatrix}$$

Chapter 5

Experimental Results

In each section, we present in details the casual DAGs along with corresponding action matrix achieved by the methodology designed in the three environments. In order to obtain a more intuitive illustration on the experimental results, for each environment, explanations generated according to [7] are also analysed.

5.1 Taxi Problem

Threshold $H = 0.1$

DAG with Action Matrix generated under different threshold H		
Causal DAG	Action Matrix	Score Matrix
$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 6 & 0 \\ 0 & 0 & 0 \\ 5 & 5 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0.597 & 0.655 & 0 \end{bmatrix}$

From the experimental result above, we can see all the edges in the causal DAG have scores over 0.5, which means all edges generated by our algorithm are highly reliable.

A illustration of the DAG along with the action matrix is shown in Figure 5.1. From the diagram, we can see destination location affects both taxi position and passenger position by drop-off action which logically makes sense. Taxi position has effect on passenger position by performing ”move” action. This is also meaningful but it is worth noticing that this causal relation is only true after taxi has picked up the passenger, however this condition is not presented in the causal graph generated.

Explanation generated

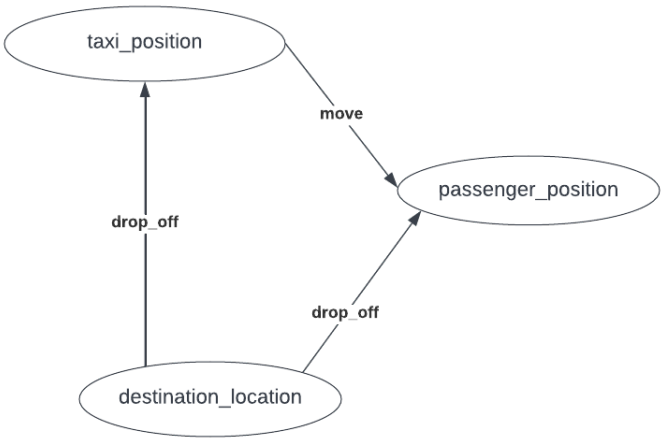

Figure 5.1: An illustration of the casual graph with action matrix in taxi problem

- ”Why?” Explanation: Before taxi pick up the passenger, taxi does ”move” action because the goal is to get passenger location.
- ”Why not?” Explanation: The taxi does action ”move”, not action ”drop off” because before taxi pick up the passenger, taxi does ”move” action because it is more desirable to do action ”move”, to change taxi position more, as the goal is to get pass location.

The explanations produced by our causal DAG and action matrix above logically make sense, which indicates that the causal DAG and it action matrix generated are rather reliable. However, there is no explanation generated with ”drop off” action, so we cannot validate the causal DAG and action matrix generated are fully correct.

5.2 CartPole Problem

Threshold $H = 0.1$

DAG with Action Matrix generated under different threshold H		
Causal DAG	Action Matrix	Score Matrix
$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 1 \\ 2 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.529 & 0 & 0.270 & 0.671 \\ 0.962 & 0 & 0 & 0.449 \\ 0.417 & 0 & 0 & 0 \end{bmatrix}$

From the experimental result above, we can see half of the edges in the causal DAG have scores over 0.5, and only one edge has score under 0.4. This indicates most edges generated by our algorithm are rather reliable.

A illustration of the DAG along with the action matrix is shown in Figure 5.2. From the diagram, we can see while most of the causal relations are logically correct, problems still exist. One problem is that, actions ”push cart right” and ”push cart left” should affect all variables in the environment equally. For example, if pole angular velocity is affected by pole angle with action ”push cart right”, it should also be affected by pole angle with action ”push cart left”, which is not shown in the diagram. The edge from pole angle to cart velocity doesn't make sense either as logically, from the description of the environment, it should always be cart that has effect on pole, as the aim of the task is to keep pole upright.

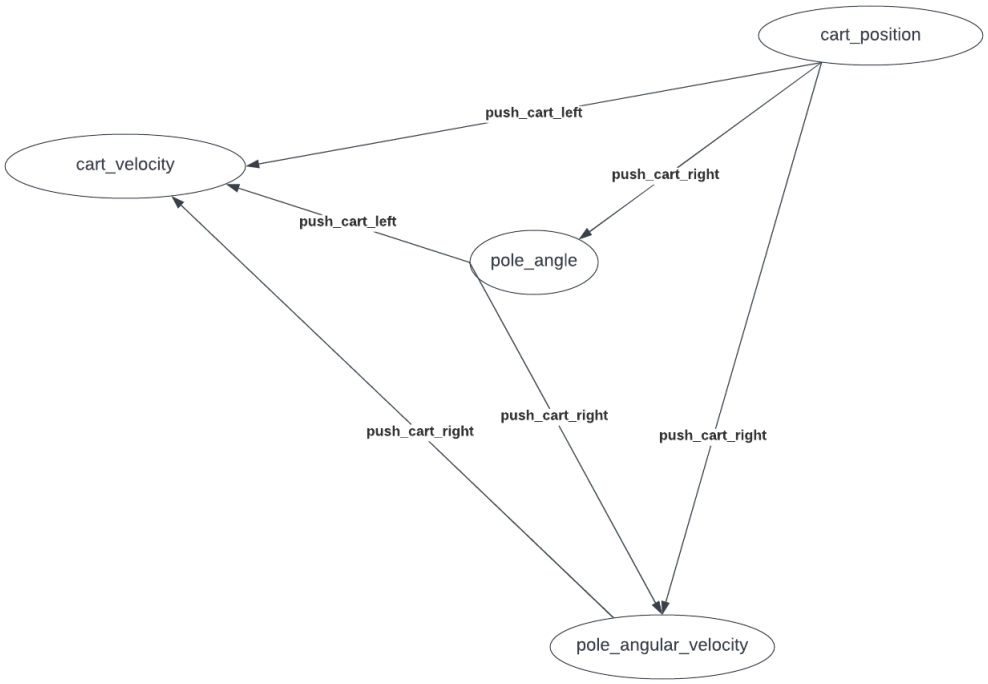

Figure 5.2: An illustration of the casual graph with action matrix in CartPole problem

Explanation generated

- ”Why?” Explanation: There are two ”Why?” explanations generated:
  1. The force pushes cart to the right because the goal is to change cart position that depends on pole angular velocity.
  2. The force pushes cart to the right because the goal is to change cart position that depends on pole angle.
- ”Why not?” Explanation: There are two ”Why not?” explanations generated:
  1. The force pushes cart to the left, but not right because it is more desirable to do action ”push cart to the left”, to have more ”pole angular velocity”, as the goal is to have ”cart position”.

2. The force pushes cart to the left, but not right because it is more desirable to do action ”push cart to the left”, to have more ”pole angular velocity”, as the goal is to have better ”cart position”.

The explanations produced based on our causal DAG and action matrix above logically make sense, which verifies the causal DAG and it action matrix generated. It is worth noticing that the cart position decides if the pole is balanced or not, so the goal: have better ”cart position” is reasonable. The causal relation between cart velocity and cart position is not shown in the explanations above, so we cannot fully verify the DAG and action matrix produced.

5.3 LunarLand Problem

Threshold $H = 0.1$.

DAG with Action Matrix generated under different threshold H

Causal DAG

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Action Matrix

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 5 & 5 & 0 & 0 & 0 & 0 \\ 5 & 2 & 2 & 0 & 0 & 0 \\ 2 & 2 & 2 & 5 & 0 & 0 \\ 5 & 5 & 2 & 5 & 5 & 0 \end{bmatrix}$$

Score Matrix

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0.684 & 0 & 0 & 0 & 0 & 0 \\ 0.565 & 0.598 & 0 & 0 & 0 & 0 \\ 0.592 & 0.679 & 0.721 & 0 & 0 & 0 \\ 0.606 & 0.704 & 0.902 & 0.632 & 0 & 0 \\ 1 & 0.979 & 0.657 & 0.650 & 0.719 & 0 \end{bmatrix}$$

From the experimental result above, we can see again, all the edges in the causal DAG have scores over 0.5, which means all edges generated by our algorithm are highly reliable.

A illustration of the DAG along with the action matrix is shown in Figure 5.3. As we can see, this diagram is quite complex with a great number of edges. This graph is partially correct. The main problem of this causal graph is that it interprets correlation as causality. For example, there is correlation between x-coordinate and y-coordinate as while the value of y-coordinate changes, clearly the value of x-coordinate also changes. However, y-coordinate is not the factor that affects the value of x-coordinate, but the velocity of lander in x-axis indeed has effect on the value of x-coordinate.

Explanation generated

- ”Why?” Explanation: Under different situations that depends on the lander’s current position, there are three explanations on why perform ”do nothing” action:

Figure 5.3: An illustration of the casual graph with action matrix in LunarLand problem

1. Because the goal is to change the coordinate of the lander in x, Which is influenced by its linear velocities in x, the coordinate of the lander in y, that depends on its linear velocities in y.
2. Because the goal is to change the coordinate of the lander in x.
3. Because the goal is to change the coordinate of the lander in x, that depends on the coordinates of the lander in y.

Similarly, there are also three explanations on why perform ”fire main engine” action:

1. Because the goal is to increase the coordinate of the lander in x-axis.
2. Because the goal is to increase the coordinate of the lander in x-axis that depends on the coordinate of the lander in y-axis.
3. Because the goal is to increase the coordinate of the lander in x-axis that depends on its linear velocities in x-axis.

We can tell that some explanations produced above are not correct due to the incorrectness of the causal DAG and action matrix generated. For example, because we interpret correlation between x-coordinate and y-coordinate to casual relation, the third explanation of why perform ”do nothing” action and the second explanation of why perform ”fire main engine” action do not make any sense logically.

- ”Why not?” Explanation: Under different situations that depends on the lander’s current position, there are two explanations on why perform ”do nothing” action but not action ”fire main engine”:

  1. Because it is more desirable to do action do nothing to have more the coordinate of the lander in y-axis, in order to have more its linear velocity in x-axis, as the goal is to have the coordinate of the lander in x-axis.
  2. Because it is more desirable to do action do nothing to have more its linear velocity in x-axis, as the goal is to have the coordinate of the lander in x-axis.

  Similarly, there are also three explanations on why perform ”fire main engine” action but not action ”do nothing”:

  1. Because it is more desirable to do action fire main engine to have less its linear velocity in y-axis in order to have more its angle, as the goal is to have the coordinate of the lander in x-axis.
  2. Because it is more desirable to do action fire main engine, to have more the coordinate of the lander in y-axis, as the goal is to have the coordinate of the lander in x-axis.
  3. Because it is more desirable to do action fire main engine, to have more its linear velocity in x-axis, to have less its linear velocity in y-axis, as the goal is to have the coordinate of the lander in x-axis.

  As to ”why not” explanations generated, most of them are reasonable, which to some extent prove the causal graph along with the action matrix we produced is correct. But still, some of them are logically wrong. For example, in the first explanation of why not perform ”fire main engine ” action, having more y-coordinate of the lander cannot increase its velocity in a-axis intuitively. Also in the second explanation of why not perform ”do nothing” action, there is no causality between the lander’s x and y coordinates.

Chapter 6

Evaluation

From the experimental results above, it is clear that for a RL agent, by the method designed, the causal DAG along with its action matrix can be generated successfully and meaningful explanations then can be produced as well.

However, there are still some problems worth discussing:

- We chose environments with different state spaces. By comparing the results from different environments, we can see the DAG generated of the environment with fewer variables could be more reliable. For example, the casual graph generated in taxi problem is more logically correct than the other two.
- Our designed algorithm could treat correlation between two variables as causal relation. This is mentioned in section 5.3 and further work needs to be studied to explore some way solving this problem.
- Although in chapter 5, for each environment, explanations are generated and we have evaluated the causal DAG along with its action matrix from logical aspects, a computational evaluation is still so our algorithm can be validated in a more quantitative way.

Chapter 7

Future work

This project aims to infer the causal DAG along with its action matrix of a RL environment simultaneously. While the goal of this project has been basically met, based on the evaluations in chapter 6, the following parts can still be further explored.

The rest of this chapter is divided into two sections. In section 7.1, we discuss the possibility of distinguishing correlations from causality to improve the inferred causal DAG by training a RL agent to perform interventions through meta-reinforcement Learning. In section 7.2, a computational approach is introduced that can do quantitative evaluation on the method we designed.

7.1 Causal Discovery by Meta-reinforcement Learning and Interventions

The main idea is to train a RL agent by taking meta-RL method with model-free RL. The trained agent is then able to infer causality in new environments by targeting to get rewards. Since the trained agent here can make causal inferences from observational data with informative intervention selection [29], we can apply the agent to do interventions (introduced in section 2.2.4)in our environments, so that correlations can be distinguished from causal relations.

The approach of meta-learning is to learn the learning (or inference/estimation) procedure itself, directly from data [13]. When learning new environments or doing new tasks, humans tend to use knowledge from previous experience and tend to focus on applying the methods that are likely to be implemented in the tasks. Meta-learning try to apply this human concept into machine learning process so that agents can learn to solve tasks not only by observational data, but also interactions with the new environment based on previous knowledge.

Meta-RL is about applying meta-learning to a RL environment. In the method we proposed, we treat the whole learning algorithm as a recurrent neural network (RNN). The weights of the RNN is trained with model-free RL on a wide range of learning problems [30]. In this way, the RNN is trained to learn an algorithm that is able to solve new learning problems same as or simlilar to the training problems. Learning the RNN can be split into two parts: "outer loop" of learning and "inner loop" of learning. The outer loop learns the weights of the RNN which will be used to form an "inner loop" algorithm. This inner loop learning algorithm is then used to encode the activation dynamics of the RNN and it will keep on learning when the RNN weights are frozen.

The training process can be divided into information phase and quiz phase and is briefly introduced as follows [29]: In each episode with T steps, the information phase contains the first $T-1$ steps and the quiz phase corresponds to the final step T. The "inner loop" of learning occurs in information phase. Agent in information phase interacts with the environment or obtain observed data in passive ways to get information needed. Then in quiz phase, the agent will use the causal knowledge it collected in the information phase and perform external interventions.

7.2 Computational Evaluation Approach

Based on [7], a computational evaluation approach is provided by Algorithm 5.

Algorithm 5 Task Prediction: Action Influence Model

Require: trained regression model $\mathcal{L}$, current state S_t
Ensure: predicted action α
1: $\boldsymbol{F_p} \leftarrow []$; ▷ vector of predicted difference
2: **for** every $\hat{L} \in \mathcal{L}$ **do**
3: $P_y \leftarrow \hat{L} \cdot predict(S_{x.t})$; ▷ predict variable S_y at S_{t+1}
4: $\boldsymbol{F_p} \leftarrow |S_y - P_y|$; ▷ difference with actual S_y value
5: **end for**
6: return $max(\boldsymbol{F_p}) \cdot getAction()$

The main idea here is to check if the next action of the agent can be predicted accurately enough by comparing the action predicted in the algorithm with the agent's actual next action. If not, then action influence model generated from the causal DAG with the action matrix is not accurate, which indicates the the causal DAG and the action matrix inferred is not reliable.

In Algorithm 5, we first get the predicted variable values of the next state S_{t+1} by instantiating all the action influence equations (every $\hat{L}$) in the action influence model (regression model $\mathcal{L}$) with the values of current agent's state $S_{x.t}$ (line 3). As the RL agent will always try to follow the optimal policy, the action that has the biggest

impact on correcting the policy will always be performed. The impact of an action is measured by the absolute difference between the value of predicted state variable and the value of actual state variable [7]. So we find out the action influence equation which gives the predicted value of the next state variable that has the maximum difference with the actual one and the action associated with this equation is the next action predicated to be performed.

Chapter 8

Conclusion

Explainability in reinforcement learning is a relatively recent subject of research that has developed rapidly as a result of the rising number of areas that RL agents have been applied to and the requirement for humans to comprehend them so that their potentials can be fully discovered. This project focuses on causal discovery in explainable reinforcement learning, which is a field drawing more and more attention these days.

Overall, the purpose of this project is to generate explanations for how a RL agent comes to its decisions. The main contribution of this project is to strengthen the framework described in [7] within this goal. In this framework, with the casual DAG and the action matrix of the environment provided manually, an action influence model of the agent in this environment can be generated automatically. Then under the agent's current state in the environment, explanations of "why" or "why not" the agent takes certain action can be generated. The whole framework would be complete if the agent could infer the DAG along with the action matrix of the environment by itself automatically, which becomes the main focus of my research.

The agent in this framework is given a DAG along with a action matrix of the environment manually, and used these to develop an action influence model. It was able to provide explanations for the acts it performed and did not take using this action impact model by producing instantiations and counterfactual situations. The framework would be complete if the agent could infer the DAGs and action matrix of the environment on its own, therefore causal discovery became the focus of my research.

The whole idea of the algorithm I designed is based on the GES algorithm with BIC score, which is an existing score-based causal discovery methods to RL environments. The outcome produced by my algorithm is rather satisfactory and meaningful explanations were generated by applying the DAG with action matrix into the framework in [7].

However, since the study on causal discovery in reinforcement learning is still on its early stage and interpreting the action matrix along with its causal DAG has never been done simultaneously, the algorithm still has space to be developed further in order to find out a completely functional solution that leads to fairly accurate results. Also computational evaluation are needed to fully evaluate the algorithm. In this case, further research directions in these two areas have been identified and detailed explanations are given as well.

Chapter 9

Ethical Considerations

After discussing with my tutor on the ethical checklist provided by computing department, I confirmed that there are no ethical considerations in my project. Specific aspects are reviewed as following:

1. Humans: No human participants are involved in this project
2. Protection of Personal Data: No personal data is collected in this project.
3. Animals: No animals are involved in this project
4. Developing Countries: No developing countries are involved in this project
5. Environmental Protection and Safety: This project is irrelevant to environmental aspects.
6. Dual Use and Misuse: This project have no potential for military application or human abuse.
7. Legal Issues: This project is irrelevant to software having copyright licensing implications.
8. Other Ethics Issues: This project involves no other ethics issues.

Bibliography

[1] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. MIT press, 2018. pages 1, 3, 7, 9

[2] David Maxwell Chickering. Learning Bayesian Networks is NP-Complete, pages 121–130. Springer New York, New York, NY, 1996. ISBN 978-1-4612-2404-4. doi: 10.1007/978-1-4612-2404-4_12. URL https://doi.org/10.1007/978-1-4612-2404-4_12. pages 1

[3] Andrew D Selbst and Julia Powles. Meaningful information and the right to explanation. International Data Privacy Law, 7(4):233–242, 12 2017. ISSN 2044-3994. doi: 10.1093/idpl/ipx022. URL https://doi.org/10.1093/idpl/ipx022. pages 1

[4] Erika Puiutta and Eric MSP Veith. Explainable reinforcement learning: A survey, 2020. URL https://arxiv.org/abs/2005.06247. pages 1

[5] Steven Sloman. Causal models: How people think about the world and its alternatives. Causal Models: How People Think about the World and Its Alternatives, pages 1–211, 01 2007. doi: 10.1093/acprof:oso/9780195183115.001.0001. pages 1, 10

[6] Michael Waldmann, York Hagmayer, and Aaron Blaisdell. Beyond the information givencausal models in learning and reasoning. Current Directions in Psychological Science - CURR DIRECTIONS PSYCHOL SCI, 15:307–311, 12 2006. doi: 10.1111/j.1467-8721.2006.00458.x. pages 1

[7] Suraj Nair, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Causal induction from visual observations for goal directed tasks, 2019. URL https://arxiv.org/abs/1910.01751. pages 1, 2, 10, 11, 17, 30, 38, 39, 40

[8] Biran and Courtenay V. Cotton. Explanation and justification in machine learning : A survey. 2017. pages 5, 7

[9] Abdulrahman A. Ahmed, Ayman Ghoneim, and Mohamed Saleh. Optimizing stock market execution costs using reinforcement learning. In 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pages 1083–1090, 2020. doi: 10.1109/SSCI47803.2020.9308153. pages 7

[10] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018. pages 8

[11] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. Advances in neural information processing systems, 12, 1999. pages 9

[12] Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach — part 1: Causes, 2013. URL https://arxiv.org/abs/1301.2275. pages 10

[13] Joaquin Vanschoren. Meta-learning: A survey, 2018. URL https://arxiv.org/abs/1810.03548. pages 10, 37

[14] Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach. part ii: Explanations, 2002. URL https://arxiv.org/abs/cs/0208034. pages 10

[15] Tom S. Verma and Judea Pearl. On the equivalence of causal models, 2013. URL https://arxiv.org/abs/1304.1108. pages 11, 13

[16] Norman Biggs. Algebraic Graph Theory. Cambridge Mathematical Library. Cambridge University Press, 2 edition, 1974. doi: 10.1017/CBO9780511608704. pages 12

[17] Yangbo He, Jinzhu Jia, and Bin Yu. Counting and exploring sizes of markov equivalence classes of directed acyclic graphs. Journal of Machine Learning Research, 16(79):2589–2609, 2015. URL http://jmlr.org/papers/v16/he15a.html. pages 13

[18] Markus Kalisch, Martin Mächler, Diego Colombo, Marloes H. Maathuis, and Peter Bühlmann. Causal inference using graphical models with the r package pcalg. Journal of Statistical Software, 47(11):1–26, 2012. doi: 10.18637/jss.v047.i11. URL https://www.jstatsoft.org/index.php/jss/article/view/v047i11. pages 13

[19] Clark Glymour, Kun Zhang, and Peter L. Spirtes. Review of causal discovery methods based on graphical models. Frontiers in Genetics, 10, 2019. pages 13, 14, 15

[20] Peter Spirtes, Clark N Glymour, Richard Scheines, and David Heckerman. Causation, prediction, and search. MIT press, 2000. pages 14

[21] Michail Tsagris. Bayesian network learning with the pc algorithm: An improved and correct variation. Applied Artificial Intelligence, 33(2):101–123, 2018. doi: 10.1080/08839514.2018.1526760. URL https://www.researchgate.net/profile/Michail_Tsagris/publication/

327884019_Bayesian_Network_Learning_with_the_PC_Algorithm_An_Improved_and_Correct_Variation/links/5bab44c945851574f7e65688/Bayesian-Network-Learning-with-the-PC-Algorithm-An-Improved-and-Correct-Variati pdf. pages 15

[22] David Maxwell Chickering. Optimal structure identification with greedy search. J. Mach. Learn. Res., 3(null):507–554, mar 2003. ISSN 1532-4435. doi: 10.1162/153244303321897717. URL https://doi.org/10.1162/153244303321897717. pages 16

[23] Alain Hauser and Peter Bühlmann. Characterization and greedy learning of interventional markov equivalence classes of directed acyclic graphs. 2011. doi: 10.48550/ARXIV.1104.2808. URL https://arxiv.org/abs/1104.2808. pages 17

[24] Ernst Wit, Edwin van den Heuvel, and Jan-Willem Romeijn. 'all models are wrong...': an introduction to model uncertainty. Statistica Neerlandica, 66(3): 217–236, 2012. pages 17

[25] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. arXiv preprint arXiv:1606.01540, 2016. pages 23

[26] simoninithomas. https://github.com/simoninithomas/Deep_reinforcement_learning_Course/, 2018. pages 25

[27] goodboychan. https://github.com/goodboychan/goodboychan.github.io/blob/main/_notebooks/2021-05-12-REINFORCE-CartPole.ipynb, 2021. pages 26

[28] goodboychan. https://github.com/goodboychan/goodboychan.github.io/blob/main/_notebooks/2021-05-07-DQN-LunarLander.ipynb, 2021. pages 28

[29] Ishita Dasgupta, Jane X. Wang, Silvia Chiappa, Jovana Mitrovic, Pedro A. Ortega, David Raposo, Edward Hughes, Peter W. Battaglia, Matthew M. Botvinick, and Zeb Kurth-Nelson. Causal reasoning from meta-reinforcement learning. CoRR, abs/1901.08162, 2019. URL http://arxiv.org/abs/1901.08162. pages 37, 38

[30] Daniel G. McClement, Nathan P. Lawrence, Philip D. Loewen, Michael G. Forbes, Johan U. Backström, and R. Bhushan Gopaluni. A meta-reinforcement learning approach to process control. IFAC-PapersOnLine, 54(3):685–692, 2021. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2021.08.321. URL https://www.sciencedirect.com/science/article/pii/S2405896321010958. 16th IFAC Symposium on Advanced Control of Chemical Processes ADCHEM 2021. pages 38