Naveen Srinivasan
Submission for Completion of Sc.M. Requirements for Brown University Master's in CS
Spring 2020

Causal Inference for Planning in Reinforcement Learning: Part 2

1. Abstract (Copied from Part 1)

We sought to develop environment-understanding and planning methodology for two reinforcement learning (RL) environments: GridWorld and Taxi[1] using causal inference. In this paper, we propose an algorithm for learning non-confounded relationships between objects in the environment, generating actionable rules and a structural causal model (SCM) from these observations, and planning using these rule sets. We show that we are able to successfully generate rules representative of the true, non-confounded relationships between objects and rewards in the environment, and use these rules to optimize agent behavior in these domains. In future work, we hope to scale these techniques to real-world datasets and assess the efficacy of SCM-based predictions at mitigating bias when compared to standard correlatory models.

2. Background and Introduction

For Part One of our exploration, we deconfounded the reward field in a basic GridWorld environment; this component is concerned with factorizing the transition dynamics of a reinforcement-learning environment via a Structural Causal Model (SCM). For this exploration, we utilized the OOMDP variant of Dietterich's Taxi domain proposed by Diuk et al.[2]. The Taxi domain is a grid-world domain, where the taxi agent has to pick up a passenger from a start location and drop this passenger off at their respective goal destination. The action set for the taxi agent is North, East, South, West, Pickup, and Dropoff. Walls are introduced into the grid to inhibit the taxi's movement.

We experimented using a modified version of Diuk's OOMDP Taxi domain that contained three object classes: Taxi, Passenger, and Wall. Taxi, Passenger, and Wall have attributes x and y, which define their current grid location. Passenger also has a Boolean attribute in-taxi, which specifies whether the passenger is inside the taxi, in addition to attributes dest_x

---

[1] "Download PDF - arXiv." https://arxiv.org/pdf/cs/9905014. Accessed 11 May. 2020.
[2] "An Object-Oriented Representation for Efficient Reinforcement Learning." http://carlosdiuk.github.io/papers/OORL.pdf. Accessed 11 May. 2020.
[3] "Active Learning of Dynamic Bayesian Networks in Markov Decision Processes". https://link.springer.com/chapter/10.1007/978-3-540-73580-9_22. Accessed 11 May. 2020.

and dest_y, which define the grid location of this passenger's goal destination. The underlying state in an OOMDP is the union of the states of its objects. Two objects in an OOMDP define a relation when they interact with each other. We define seven relations, four of which operate with respect to the Taxi & Wall (wall-north, wall-east, wall-south, wall-west) objects, and three of which operate with respect to the Taxi & Passenger objects (on-passenger, on-destination, in-taxi). It is important to note that the on-destination Boolean relation corresponds to the Taxi being at the grid location of the passenger's destination. Using these relations, we can convert our full state representation (concatenation of attributes of all the objects) into a compact relation predicate vector of seven Booleans. This relation predicate vector provides utility for forming the nodes of a graphical model, which we will use to transparently factor our transition dynamics. The relation predicate vector of a given state and an action induces an effect, a change of value in some of the objects attributes. We defined eight effects a priori. Four of the effects are translational effects related to the Taxi object: MOVE NORTH, MOVE EAST, MOVE SOUTH, and MOVE WEST. Three of the effects are situational effects reflecting the status of the passenger object: HAILING TAXI (the passenger is not in-taxi and not on their goal destination), IN TAXI (the passenger is in-taxi), and ON DESTINATION (the passenger is not in-taxi and is on their goal destination). We also included NO EFFECT, which indicates that a given action did not change any values in the object's attributes (i.e., the current state is equal to the next state).

One common factored state representation for OOMDPs uses Dynamic Bayesian Networks (DBNs) to indicate how state variables interact with one another. There exists a rich literature in factoring the transition dynamics of MDPs using DBNs; in particular, we took inspiration from the work of Jonsson et al.[3]. regarding active learning of DBNs. In this work, they use a structure learning paradigm where they build up a DBN by inducing their agent to actuate in a way to maximize the total entropy of distributions used to evaluate potential refinements of the networks. Although learning a Bayesian network that maximizes the Bayesian Information Criterion (BIC) is generally NP-complete, their algorithm was able to efficiently converge to a ground truth DBN in several environments. Unfortunately, DBNs have the capacity to learn spurious correlations between state variables; the existence of confounders in our observational space can lead to erroneous inference. We depart from using DBNs to factor the transition dynamics of MDPs, and instead factor the transition dynamics via a SCM, which allows us to prune spurious relationships existing between state variables and prescribe how an agent's actuation induces a change in the environment.

3. Methodology: Taxi

We began our experiments with a 6x6 Taxi domain instance, in which we are trying to train our Taxi agent to pick up the Passenger and drop the Passenger off at their goal destination (which terminates the episode). Sixteen Wall objects form the border of the gridworld. The Taxi

object, the Passenger object, and one additional Wall object are randomly initialized into non-coinciding positions in the interior section of the gridworld.

Instead of directly maximizing a statistical criterion for the purpose of model refinements, we instead incentivize our agent to collect experience for each relational predicate vector setting and each action. The details of this data collection are described in Algorithm 1. We use this collected data to define the functional relationships modeled by the edges in our SCM. In this exploration, we limited the function class to the set of Boolean functions limited to conjunctive forms of size two. That is, for each (action, effect) tuple for which we have accumulated experience, we learn a Boolean function that defines what relation predicate setting allows a given action to induce a given effect. We initialize the set of potential Boolean functions to all pure literals (wall-north, not wall-north, etc). In the deterministic setting, one piece of experience that does not satisfy the Boolean formula for a given (action, effect) tuple is a sufficient condition for eliminating that formula from our set of candidate formulas. Ultimately, the resulting formulas allow us to define an SCM for each (action, effect) tuple.

Algorithm 1

Algorithm Constants:

(int) k - the cardinality of sets of factors within predicate vectors that are 'on' for a particular predicate vector (also maximum cardinality of factor sets considered)

(int) m - the number of visits to a particular (predicate, action) pair after which we are convinced of the causal nature of the outcome

Outputs: Map (predicate vector, action) → (outcomes, reward)

A list of rules mapping factor subset and action to outcome and reward

1. Generate all assignments of predicate vectors so that [0, k] bits are set to True. We propose an input of k = 2, based on the rules we implemented for our planning experiment.
    a. 0 - all are set to False
    b. 1 - only one is set to True
    c. 2 - two are set to True (all combinations)
2. (Dynamics) Maintain a mapping between all combinations of enumerated predicate vectors, actions → outcome, reward.
   (Visits) Maintain a mapping between all combinations of enumerated predicate vectors, actions → number of visits. This mapping will be used for 'active learning.'
    a. Initialize all mappings with value 'None'
    b. For all actions:

i. Enumerate the predicate vectors from the current state (predicate vector, action) combinations from current state.
ii. Determine which has been visited least frequently among those visited no more than *m* times. If all have been visited *m* times, choose the one that maximizes value. Ties will be broken using random selection.
iii. Return this action as *a*.

c. Take a step in the environment with action *a*, record reward *r*, and outcome:
i. Generate predicate vectors for current state, outcomes for next_state.
ii. If the value for (current state, *a*) is None, and not 'eliminated rule,' initialize in the dictionary with value (next state, *r*, outcome) (Dynamics table).
iii. If we have the same value, maintain in dictionary
iv. If we have a different value, store value as 'eliminated rule'
v. Increment the visit count of this (predicate vector, action) key.

3. Rule Generation
a. For each (action, effect) tuple:
i. If there exists experience:
1. Generate all pure literal candidate Boolean functions
a. For each candidate function:
i. Loop through all experience and remove the candidate function it is not satisfied by any datum of experience

1. Rule Generation
a. for each position in the feature vector:
i. if all confidence intervals generated for feature *i* overlap, generate a rule associating *i* with the mean reward found when *i* is set to 1.

4. Results

This section contains two figures. The edges express logical relations. Green edges take the setting of the relational predicate and red edges negate the setting of the relational predicate. All logical formulae projecting into a node are logically AND-ed together. The first figure is an SCM that defines the transition dynamics with respect to the (up, MOVE UP) tuple. The second figure is an SCM that defines the transition dynamics with respect to the (pickup, IN TAXI) tuple. These two examples are discussed further in the discussion section.

Figure 1: SCM for (up, MOVE UP)

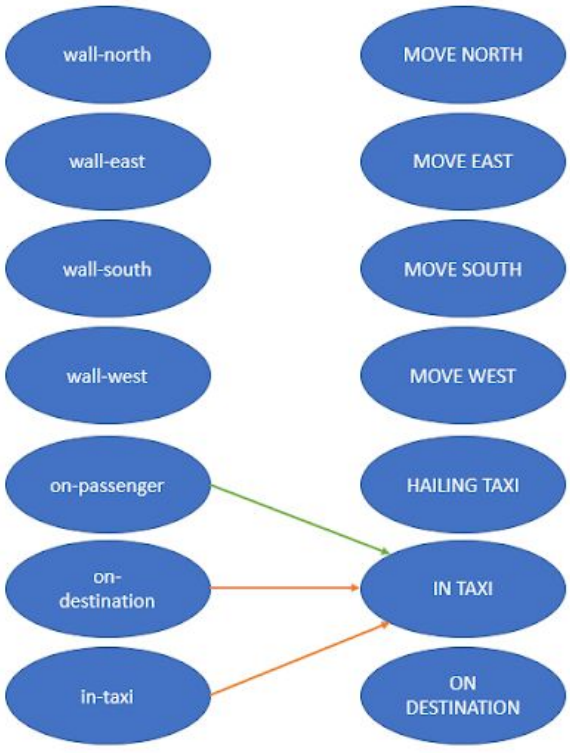

Figure 2: SCM for (pickup, IN TAXI)

5. Discussion

We have shown that our causal inference methodology factors the transition dynamics of an MDP domain into an SCM. As visible in Figure 1, it is able to eliminate any spurious dependencies between factors like in-taxi, on-destination, etc. and elucidate that the action *up* will cause our agent to MOVE UP if and only if there does not exist a wall above the agent. Figure 2 demonstrates one of the challenges of this approach. The true logical rule for the (pickup, IN TAXI) tuple is on-passenger AND NOT in-taxi, however, our agent learned the rule on-passenger AND NOT on-destination AND NOT in-taxi. In this particular case, our agent was unable to accumulate experience to eliminate NOT on-destination from our final rule. This error is due to the structure of the environment itself. Since a passenger being on destination is the termination condition for an instance, our agent will never be able to accumulate the experience it needs to eliminate this rule. It may be possible to eliminate this issue by further manipulating the training environment.

6. Future Work

One potential future task would be to use the joint experience from a given (action, effect) tuple and the respective (action, no effect) tuple to generate rules more efficiently. For example, if I observe that the *up* action yields no effect when there is wall-above, this fact provides a useful signal for understanding that the *up* action yields MOVE UP when there is NOT wall-above. In this case, the rule learned for the NO EFFECT case for the action *up* is the logical negation of the MOVE UP case. This pattern doesn't hold as cleanly with respect to our non-translational (compound) effects. For example, the rule for (pickup, no effect) is NOT on-passenger. Although the rule for (pickup, IN TAXI) does contain the logical negation of the NO EFFECT rule, it is also dependent upon other factors.

Another potential future task would be exploring how to translate this method from the deterministic setting to the stochastic setting. In the deterministic setting, we were able to construct our SCM purely using endogenous variables that were sufficient for modeling the causal connections in our environment. In the stochastic setting, when we retrieve noisy experience for each of our translational effects, we would need to introduce exogenous variables to appropriately model the transition dynamics via an SCM. This problem is theoretically interesting, although it's unlikely that the resultant SCM would provide any transparency/inference utility as compared to a DBN in the stochastic setting.

A third potential future task is further formalizing learning the correct number of conditions for a rule. In the deterministic setting, one piece of evidence that did not satisfy a rule condition was sufficient to eliminate that rule from our set of candidate rules. The final rule could be constructed via a conjunction of the surviving rules. In the stochastic setting, we can obtain evidence that contradicts a rule condition, even if this rule would be present in a

ground-truth SCM. Thus, we would need to construct the notion of a belief state for each of our candidate rules. The belief state or confidence in a given rule would be an increasing function of the magnitude of the set of experiences that satisfy the rule. Initially, our candidate rules are the pure literals of our set of relational predicates. After accumulating some experience, we can order our candidate rules with respect to confidence, and branch with respect to our most confident rule(s). In other words, we can condition upon a relational predicate taking a value and construct conditional beliefs in our remaining candidate rules. This mechanism provides an iterative method for elegantly learning the correct number of conditions for a given rule. For this method to be complete, we still need to determine an appropriate cutoff threshold/stopping criterion. This algorithm is currently formulated in a batch-mode reinforcement learning setting where we have sufficiently diverse trajectories. Future work must be done to have this method operate in an online setting where the agents actuation is immediately driven by the desire to find counterevidence to candidate rules, as opposed to ensuring we have sufficient evidence with respect to every relational predicate vector setting.

7. Acknowledgements

Thank you to our advisor, Professor Michael Littman, for over a year of mentorship and support; your patience and insights have been invaluable. Thank you to the Computer Science department at Brown for five years of education, community, and growth.