Causal Dynamics Learning for Task-Independent State Abstraction

Zizhao Wang, Xuesu Xiao, Zifan Xu, Yuke Zhu, and Peter Stone
Motivation

Real-world dynamics are usually \textit{sparse}.
- The transition of each state variable only depends on a few state variables.

For example, for an environment with a robot, two doors and a clock on the wall:
Motivation

Real-world dynamics are usually \textit{sparse}.
- The transition of each state variable only depends on a few state variables.

For example, for an environment with a robot, two doors and a clock on the wall:
Motivation

But most model-based RL work uses dense dynamics models (fully-connected networks).

sparse real-world dynamics
Motivation

But most model-based RL work uses dense dynamics models (fully-connected networks).
Motivation

But most model-based RL work uses dense dynamics models (fully-connected networks).

sparse real-world dynamics VS dense dynamics model
But most model-based RL work uses dense dynamics models (fully-connected networks).

sparse real-world dynamics VS dense dynamics model
Motivation

But most model-based RL work uses dense dynamics models (fully-connected networks).

sparse real-world dynamics VS dense dynamics model
Motivation

dense dynamics model

generalizes badly
due to spurious correlation
Motivation

dense dynamics model

generalizes badly due to spurious correlation

overfit to data noise, etc
Motivation

dense dynamics model

generalizes badly due to spurious correlation

overfit to data noise, etc
Motivation

dense dynamics model

generalizes badly
due to spurious correlation

overfit to data noise, etc
Motivation

dense dynamics model

training

open

t at \( t + 1 \)

A A

testing

close

t at \( t + 1 \)

A A

overfit to data noise, etc

generates badly
due to spurious correlation
Motivation

dense dynamics model

generalizes badly
due to spurious correlation
Motivation

dense dynamics model

generalizes badly due to spurious correlation

causal dynamics learning (CDL)

only keep causal edges, robust to outliers,
Motivation

dense dynamics model

causal dynamics learning (CDL)

dense dynamics model

causal dynamics learning (CDL)

generalizes badly due to spurious correlation

only keep causal edges, robust to outliers, e.g., clock outliers won’t affect door A & B prediction
Problem Setup

\[ \langle S, A, P \rangle \]

**S**: state space (known, *high-level* variables are given)

We leave handling low-level, partially-observable state space (e.g., images) as future work.

**A**: action space (known)

**P**: transition probability (not known)
Goals

1. Learn a causal dynamics model from transition data

\[ P(s_{t+1} \mid s_t, a_t) = \prod_{i=1}^{d_s} P(s^i_{t+1} \mid \text{PA}_{s^i}) \]

\( \text{PA}_{s^i} \) are parents of \( s^i \) during the data generation process.
Goals
1. Learn a causal dynamics model from transition data
2. Split state variables into three categories

\[ S = S^c \times S^c \times S^i \]

- \( S^c \): space of **controllable** state variables
- \( S^r \): space of **action-relevant** state variables
- \( S^i \): space of **action-irrelevant** state variables
Problem Setup

Goals
1. Learn a causal dynamics model from transition data
2. Split state variables into three categories
3. Derive a state abstraction by omitting action-irrelevant state variables
Problem Setup

Goals
1. Learn a causal dynamics model from transition data
2. Split state variables into three categories
3. Derive a state abstraction by omitting action-irrelevant state variables
4. Use the abstracted causal dynamics to learn (many) downstream tasks
Bisimulation\[^1\] : bisimulation considers two states the same \( \phi(x) = \phi(x') \) if

\[
R(x, a) = R(x', a),
\]

\[
\sum_{x'' \in \phi^{-1}(s)} P(x'' | x, a) = \sum_{x'' \in \phi^{-1}(s)} P(x'' | x', a)
\]

\[^1\] Ravindran, B., 2004; Li, L., 2009
Compared to CDL,

- Bisimulation is reward-specific (applicable to limited tasks).
  
  e.g., the bisimulation abstraction learned from “opening door A” can’t be used for “opening door B.”
Related Work

Compared to CDL,

- Bisimulation is reward-specific (applicable to limited tasks).
  e.g., the bisimulation abstraction learned from “opening door A” can’t be used for “opening door B.”

![Diagram showing the difference between CDL and bisimulation]
Compared to CDL,

- Bisimulation is reward-specific and thus applicable to **limited** tasks.
  
  In contrast, CDL’s abstraction can be applied to a larger range of tasks.
Compared to CDL,

- Bisimulation is reward-specific and thus applicable to limited tasks. In contrast, CDL’s abstraction can be applied to a larger range of tasks.
Related Work

Compared to CDL,

- Bisimulation is reward-specific and thus applicable to limited tasks.
- Most bisimulation work still uses dense dynamics, leading to poor generalization.
Method

So far, the key of CDL is to learn a causal dynamics model.

\[ \mathcal{P}(s_{t+1}|s_t, a_t) = \prod_{i=1}^{d_S} \mathcal{P}(s_{t+1}^i|\text{PA}_{s_t}^i) \]
So far, the key of CDL is to learn a causal dynamics model.

$$\mathcal{P}(s_{t+1}|s_t, a_t) = \prod_{i=1}^{d_s} \mathcal{P}(s_{t+1}^i | \text{PA}_{s^i})$$

Specifically, for each state variable $s^j$, how to determine if a state variable $s^i$ is one of its parents?
Method

Key idea: determine if the causal edge $s_t^i \rightarrow s_{t+1}^j$ exists with a conditional independence test.

Skipping assumptions and proofs,
Key idea: determine if the causal edge $s_t^i \rightarrow s_{t+1}^j$ exists with a conditional independence test.

Skipping assumptions and proofs,

**Theorem 1**

If $s_t^i \perp s_{t+1}^j | \{s_t/s_t^i, a_t\}$, then $s_t^i \rightarrow s_{t+1}^j$. 
Key idea: determine if the causal edge \( s_t^i \rightarrow s_{t+1}^j \) exists with a conditional independence test.

Theorem 1

If \( s_t^i \perp\!\!\!\perp s_{t+1}^j | \{s_t^i/ s_t^i, a_t\} \), then \( s_t^i \rightarrow s_{t+1}^j \).
Method

Key idea: determine if the causal edge $s^i_t \to s^j_{t+1}$ exists with a conditional independence test.

**Theorem 1**

If $s^i_t \perp s^j_{t+1} | \{s_t / s^i_t, a_t\}$, then $s^i_t \to s^j_{t+1}$.

In other words, is $s^i_t$ needed to predict $s^j_{t+1}$?
Key idea: determine if the causal edge $s^i_t \rightarrow s^j_{t+1}$ exists with a conditional independence test.

**Theorem 1**

If $s^i_t \perp s^j_{t+1} | \{s_t^i, a_t\}$, then $s^i_t \rightarrow s^j_{t+1}$.

In other words, is $s^i_t$ needed to predict $s^j_{t+1}$?
Key idea: determine if the causal edge $s^i_t \rightarrow s^j_{t+1}$ exists with a conditional independence test.

**Theorem 1**

If $s^i_t \perp s^j_{t+1} | \{s_t / s^i_t, a_t\}$, then $s^i_t \rightarrow s^j_{t+1}$.

In other words, is $s^i_t$ needed to predict $s^j_{t+1}$?
Key idea: determine if the causal edge $s_t^i \rightarrow s_{t+1}^j$ exists with a conditional independence test.

**Theorem 1**

If $s_t^i \perp\!\!\!\perp s_{t+1}^j | \{s_t / s_t^i, a_t\}$, then $s_t^i \rightarrow s_{t+1}^j$.

In other words, is $s_t^i$ needed to predict $s_{t+1}^j$?

\[
p(s_{t+1}^j | s_t, a_t) \begin{cases} \neq \quad & p(s_{t+1}^j | \{s_t / s_t^i, a_t\}) \end{cases}
\]
Learn and predict $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{s^i/s^i\}_t, a_t)$ using generative models, but there will be $d_S^2$ models to train...
Learning $p(s^j_{t+1} | s_t, a_t)$ & $p(s^j_{t+1} | \{s / s^i \}_t, a_t)$ needs to train $d^2_S$ models. With a mask $M_j$ and an element-wise maximum module, one network can represent all generative models in the form of $p(s^j_{t+1} | \cdot)$. 
Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j | s_t, a_t)$,
Learning $p(s^j_{t+1}|s_t, a_t)$ & $p(s^j_{t+1}|\{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s^j_{t+1}|s_t, a_t)$,
Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j | s_t, a_t)$,
Learning $p(s^j_{t+1}|s_t, a_t)$ & $p(s^j_{t+1}|\{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s^j_{t+1}|s_t, a_t)$,
Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{ s/s^i \}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j | s_t, a_t)$,

$$
\begin{array}{c|cccc}
\text{inputs} & a_t & f^a \\
1 & & 1 & 0 & 2 & 0 \\
2 & s_1^i & f_1^i & 0 & 2 & 0 & 2 \\
\vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots \\
0 & s_i^i & f^i & -1 & 0 & 3 & 0 \\
\vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots \\
-1 & s_t^d & f^d & 0 & 4 & 1 & -2 \\
\end{array}
$$
Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j | s_t, a_t)$,
Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j | s_t, a_t)$,
Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{ s/s^i \}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j | s_t, a_t)$,

<table>
<thead>
<tr>
<th>inputs</th>
<th>features</th>
<th>mask</th>
<th>masked features</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_t$</td>
<td>$f^a$</td>
<td>$M^j$</td>
<td>$f^a$</td>
</tr>
<tr>
<td>1</td>
<td>1 0 2 0</td>
<td></td>
<td>1 0 2 0</td>
</tr>
<tr>
<td>$s_t^1$</td>
<td>$f^1$</td>
<td></td>
<td>$f^1$</td>
</tr>
<tr>
<td>2</td>
<td>0 2 0 2</td>
<td></td>
<td>0 2 0 2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$s_t^i$</td>
<td>$f^i$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-1 0 3 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$s_t^{d_s}$</td>
<td>$f^{d_s}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>0 4 1 -2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Learning $p(s^j_{t+1}|s_t, a_t) \& p(s^j_{t+1}|\{s/s^i\}_t, a_t)$ needs to train $d^2_S$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s^j_{t+1}|s_t, a_t)$,

- **Method**
Learning $p(s^j_{t+1}|s_t, a_t)$ & $p(s^j_{t+1}|\{s/s^i\}_t, a_t)$ needs to train $d^2_S$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s^j_{t+1}|s_t, a_t)$,
Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j | s_t, a_t),}

```plaintext

<table>
<thead>
<tr>
<th>inputs</th>
<th>features</th>
<th>mask</th>
<th>masked features</th>
<th>global feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_t$</td>
<td>$f^a$</td>
<td>$1$</td>
<td>$0$</td>
<td>$2$</td>
</tr>
<tr>
<td>$s_t^1$</td>
<td>$f^1$</td>
<td>$0$</td>
<td>$2$</td>
<td>$0$</td>
</tr>
<tr>
<td>$s_t^2$</td>
<td>$0$</td>
<td>$2$</td>
<td>$-1$</td>
<td>$3$</td>
</tr>
<tr>
<td>$s_t^{d_s}$</td>
<td>$0$</td>
<td>$4$</td>
<td>$1$</td>
<td>$-2$</td>
</tr>
</tbody>
</table>
```

```plaintext

$M_j$ and $f^a$ are applied to the features. The maximum is taken across the features to get the global feature.

$M_j = \max(f^1, f^2, \ldots, f^{d_s})$

$h_j = f^a(M_j)$

```
Learning $p(s_{t+1}^j|s_t, a_t)$ & $p(s_{t+1}^j|\{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j|s_t, a_t)$,
Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{s/s^i \}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j | s_t, a_t)$,
Learning \( p(s^j_{t+1}|s_t, a_t) \) & \( p(s^j_{t+1}|\{s/s^i\}_t, a_t) \) needs to train \( d_S^2 \) models. With \( M_j \) and an element-wise maximum module, one network can represent all models. For example, to represent \( p(s^j_{t+1}|s_t, a_t) \),
Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j | s_t, a_t)$,
Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models.

With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$,

<table>
<thead>
<tr>
<th>inputs</th>
<th>features</th>
<th>mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_t$</td>
<td>$f^a$</td>
<td>$M^j$</td>
</tr>
<tr>
<td>1</td>
<td>0 2 0 0</td>
<td></td>
</tr>
<tr>
<td>$s^1_i$</td>
<td>$f^1$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0 2 0 2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$s^i_i$</td>
<td>$f^i$</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-1 0 3 0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$s^{d_S}_i$</td>
<td>$f^{d_S}$</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>0 4 1 -2</td>
<td></td>
</tr>
</tbody>
</table>
Method

Learning $p(s_{t+1}^j | s_t^j, a_t)$ & $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$,

\[
\begin{align*}
\text{inputs} & \quad \text{features} & \quad \text{mask} & \quad \text{masked features} \\
& a_t & f^a & M^j & f^a \\
1 & 1 & 0 & 2 & 0 \\
2 & s_1 & 0 & 2 & 0 & 2 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
& s_t^i & f^i & \vdots & \vdots \\
0 & 0 & -1 & 0 & 3 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
& s_t^{dS} & f^{dS} & \vdots & \vdots \\
-1 & 0 & 4 & 1 & -2
\end{align*}
\]
Method

Learning $p(s_{t+1}^j|s_t, a_t)$ & $p(s_{t+1}^j|\{s/s_i^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j|\{s/s_i^i\}_t, a_t)$,

<table>
<thead>
<tr>
<th>inputs</th>
<th>features</th>
<th>mask</th>
<th>masked features</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_t$</td>
<td>$f^a$</td>
<td>$M^j$</td>
<td>$f^a$</td>
</tr>
<tr>
<td>1</td>
<td>1 0 2 0</td>
<td>1 0 2 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$s_i^1$</th>
<th>$f^1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0 2 0 2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$s_i^i$</td>
<td>$f^i$</td>
</tr>
<tr>
<td>0</td>
<td>-1 0 3 0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$s_i^{d_S}$</td>
<td>$f^{d_S}$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

| $-1$ | 0 4 1 -2 |
Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{i+1}^j | \{s/s^i\}_t, a_t)$,
Method

Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j \{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j \{s/s^i\}_t, a_t)$,

<table>
<thead>
<tr>
<th>inputs</th>
<th>features</th>
<th>mask</th>
<th>masked features</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_t$</td>
<td>$f^a$</td>
<td></td>
<td>$f^a$</td>
</tr>
<tr>
<td>1</td>
<td>1 0 2 0</td>
<td>1 0 2 0</td>
<td></td>
</tr>
<tr>
<td>$s_t$</td>
<td>$f^1$</td>
<td></td>
<td>$f^1$</td>
</tr>
<tr>
<td>2</td>
<td>0 2 0 2</td>
<td>0 2 0 2</td>
<td></td>
</tr>
<tr>
<td>$s_t^i$</td>
<td>$f^i$</td>
<td></td>
<td>$f^i$</td>
</tr>
<tr>
<td>$s_t^{d_s}$</td>
<td>$f^{d_s}$</td>
<td></td>
<td>$f^{d_s}$</td>
</tr>
<tr>
<td>-1</td>
<td>0 4 1 -2</td>
<td>0 4 1 -2</td>
<td></td>
</tr>
</tbody>
</table>
Method

Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$ needs to train $d_S^{2}$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{i+1}^j | \{s/s^i\}_t, a_t)$,
Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{i+1}^j | \{s/s^i\}_t, a_t)$,
Method

Learning $p(s^j_{t+1}|s_t, a_t)$ & $p(s^j_{t+1}|\{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s^j_{t+1}|\{s/s^i\}_t, a_t)$,
Learning $p(s_{t+1}^j | s_t, a_t)$ & $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models.
With $M_j$ and an element-wise maximum module, one network can represent all models.
For example, to represent $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$,
Method

Learning $p(s_{t+1}^j | s_t^j, a_t)$ & $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$ needs to train $d_S^2$ models. With $M_j$ and an element-wise maximum module, one network can represent all models. For example, to represent $p(s_{t+1}^j | \{s/s^i\}_t, a_t)$,
After training, to represent the causal model \( p(s^j_{t+1} | PA^j_t) \), we can adjust the mask to select causal parents of \( s^j \) only.
Causal Dynamics Learning (CDL)
Data collection policy

transition buffer

Learn causal dynamics

Causal Dynamics Learning (CDL)
Causal Dynamics Learning (CDL)

Data collection policy

Build the causal graph and state abstraction

Learn causal dynamics

transition buffer
Causal Dynamics Learning (CDL)

Data collection policy → transition buffer

Learn causal dynamics

Build the causal graph and state abstraction

Learn downstream tasks with the abstracted causal dynamics
Experiments

Baselines

Monolithic

MLP: multi-layer perceptron

Modular

Experiments

Baselines

MLP: multi-layer perceptron

Experiments

Baselines

Monolithic
MLP: multi-layer perceptron

Modular

Regularization[2]

Graph Neural Network[3]

regularize the number of inputs

Does each baseline learn a causal model?

MLP: multi-layer perceptron

Experiments

Chemical Environment\cite{ke2021}

Synthesized environment
- with different underlying graphs

\begin{itemize}
\item chain
\item collider
\item full
\end{itemize}

\cite{ke2021} Ke et al., Neurips 2021.
Experiments

Chemical Environment\cite{Ke2021}

Synthesized environment
- with different underlying graphs
- as action changes the color of one node, colors of all its descendants will also change.

\cite{Ke2021} Ke et al., Neurips 2021.
Experiments

Chemical Environment[4]

Synthesized environment
– with different underlying graphs
– as action changes the color of one node, colors of all its descendants will also change.

Action-irrelevant variables: positions sampled from $N(0, 0.01)$.

Experiments

Manipulation Environment

State Variables:
Experiments

Manipulation Environment

State Variables:
- end-effector (eef)
Experiments

Manipulation Environment

State Variables:
- end-effector (eef)
- gripper (grp)
State Variables:
- end-effector (eef)
- gripper (grp)
- the movable object (mov)
Experiments

Manipulation Environment

State Variables:
- end-effector (eef)
- gripper (grp)
- the **movable** object (mov)
- the **unmovable** object (unm)
Experiments

Manipulation Environment

State Variables:
- end-effector (eef)
- gripper (grp)
- the movable object (mov)
- the unmovable object (unm)
- the randomly moving object (rand)
State Variables:
- end-effector (eef)
- gripper (grp)
- the movable object (mov)
- the unmovable object (unm)
- the randomly moving object (rand)
- non-interactable markers (mkr\textsuperscript{1-3})
Experiments

Manipulation Environment

State Variables:
- end-effector (eef)
- gripper (grp)
- the movable object (mov)
- the unmovable object (unm)
- the randomly moving object (rand)
- non-interactable markers (mkr$^{1-3}$)

Action dimensions:
- end-effector target
State Variables:
- end-effector (eef)
- gripper (grp)
- the **movable** object (mov)
- the **unmovable** object (unm)
- the **randomly moving** object (rand)
- non-interactable markers (mkr\(^{1-3}\))

Action dimensions:
- end-effector target
- gripper open/close
At the object level, the learned dependence is (subjectively) reasonable.
## Results

### Causal Graph Accuracy

<table>
<thead>
<tr>
<th>Environment</th>
<th>CDL (Ours)</th>
<th>Reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chemical (Collider)</td>
<td>100.0 ± 0.0</td>
<td>99.4 ± 0.4</td>
</tr>
<tr>
<td>Chemical (Chain)</td>
<td>100.0 ± 0.1</td>
<td>99.7 ± 0.1</td>
</tr>
<tr>
<td>Chemical (Full)</td>
<td>99.1 ± 0.1</td>
<td>97.7 ± 0.4</td>
</tr>
<tr>
<td>Manipulation</td>
<td>90.2 ± 0.3</td>
<td>84.4 ± 0.5</td>
</tr>
</tbody>
</table>

*Table 1. Causal Graph Accuracy (in %) for CDL and Reg*
Results

Dynamics Generalization

Causal dynamics generalizes best in unseen states.
Results

Dynamics Generalization

Causal dynamics generalizes best in unseen states.

- Causal Dynamics Learning (Ours)
- Regularization
- Graph Neural Network
- Modular
- Monolithic

ID: in-distribution states
Causal dynamics generalizes best in unseen states.

**Results**

Causal dynamics learning (Ours)

Regularization

Graph Neural Network

Modular

Monolithic

ID: in-distribution states

OOD: out-of-distribution states
Results

Causal dynamics generalizes best in unseen states.

**Task Generalization**

ID: in-distribution states

OOD: out-of-distribution states
Limitations and Future Directions

Scale to high-dimensional observations (e.g. images)?
- Learn disentangled representations, then learn dynamics in the representation space

Causal dependencies are learned globally only.
- Learning local independencies to further sparsify the dynamics.
Causal Dynamics Learning for Task-Independent State Abstraction

Zizhao Wang, Xuesu Xiao, Zifan Xu, Yuke Zhu, and Peter Stone

Contact Information:
Zizhao Wang: zizhao.wang@utexas.edu

CDL’s **state abstraction** omits action-irrelevant variables.

What tasks can this state abstraction solve?

- **✓** Tasks whose rewards are defined by **controllable** and **action-relevant** state variables
- **✗** Tasks with rewards involving **action-irrelevant** state variables

Solving any task (learning any reward) means no abstraction.
Method

Key idea: determine if the causal edge $s^i_t \rightarrow s^j_{t+1}$ exists with a conditional independence test.

Theorem 1

If $s^i_t \perp\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!
\!