Problem Set 3

6.S091: Causality

IAP 2023

Due: Friday, February 3rd at 11:59pm EST, by email (**csquires@mit.edu**)

- Problem sets **must** be done in LaTeX.
- You may use any programming language for your solutions, and you are not required to turn in your code.

Note: while there are a total of 15 points on this pset, 10 will still be considered a perfect score. Anything above 10 will be "bonus" points and will count toward the 18 points needed to pass.

Problem 1: Constructing Minimal I-MAPs [5 points]

You should use the partial-correlation based conditional independence test which you coded in Problem Set 2. An implementation of the necessary functions can be found at

`https://github.com/csquires/6.S091-causality/blob/main/psets/pset3/utils.py`

(a) Write a function

`minimal_map(samples, permutation, alpha)`

which constructs the graph $\mathcal{G}_\pi$ for the `permutation` π, using the partial-correlation based conditional independence test with significance level `alpha`.

Samples of $(X_1, X_2, X_3, X_4, X_5)$ are in the file `imap_samples.csv` at

`https://github.com/csquires/6.S091-causality/blob/main/psets/pset3/imap_samples.csv`

Run `minimal_imap(imap_samples, permutation, 0.05)` for the following permutations:

$$\pi_a = [1, 2, 3, 4, 5]$$
$$\pi_c = [5, 4, 1, 2, 3]$$
$$\pi_b = [5, 4, 3, 2, 1]$$

The graph $\mathcal{G}_{\pi_a}$ should have 4 edges. Draw the graphs for each permutation π_a, π_b, and π_c.

(b) Recall that a Chickering sequence from $\mathcal{G}$ to $\mathcal{H}$ is a sequence of DAGs, where each consecutive pair in the sequence is related by a covered edge reversal or an edge addition. Construct a Chickering sequence from $\mathcal{G}_{\pi_a}$ to $\mathcal{G}_{\pi_b}$. You can do this problem by hand, you do not need to write a function.

Problem 2 : d-separation defines a graphoid [5 points]

In this problem, you will show that two of the graphoid properties hold for d-separation. Throughout the problem, let $\mathbf{A}, \mathbf{B}_1, \mathbf{B}_2, \mathbf{S} \subseteq \mathcal{X}$ be disjoint subsets of nodes in a DAG $\mathcal{G}$, and let $\mathbf{B} = \mathbf{B}_1 \cup \mathbf{B}_2$.

(a) Weak Union. Suppose there is a d-connecting path γ from $A \in \mathbf{A}$ to $B \in \mathbf{B}_1$ given $\mathbf{S} \cup \mathbf{B}_2$. Show that there is a d-connecting path γ' from some $A' \in \mathbf{A}$ to some $B' \in \mathbf{B}$ given $\mathbf{S}$.

(b) Contraction. Suppose there is a d-connecting path γ from $A \in \mathbf{A}$ to $B \in \mathbf{B}$ given $\mathbf{S}$. Show that either:

- There is a d-connecting path γ' from some $A' \in \mathbf{A}$ to some $B' \in \mathbf{B}_2$ given $\mathbf{S}$, or
- There is a d-connecting path γ' from some $A' \in \mathbf{A}$ to some $B' \in \mathbf{B}_1$ given $\mathbf{S} \cup \mathbf{B}_2$.

Problem 3: Searching an Equivalence Class [5 points]

The folder

`https://github.com/csquires/6.S091-causality/tree/main/psets/pset3/mec_examples`

contains pairs of the form

`starting_dag1.csv`   `ending_dag1.csv`

where the starting DAG $\mathcal{G}$ is Markov equivalent to the ending DAG $\mathcal{H}$. Each file represents a DAG in adjacency matrix form, i.e., each file stores a matrix A where $A_{ij} = 1$ if and only if the associated DAG $\mathcal{G}$ has an edge $i \to j$. Starter code for loading DAGs can be found at

`https://github.com/csquires/6.S091-causality/blob/main/psets/pset3/search_mec.py`

(a) Write a function

`covered_edge_neighbors(dag)`

which, given a DAG $\mathcal{G}$, returns all DAGs $\mathcal{G}'$ which differ from $\mathcal{G}$ by exactly one covered edge reversal. `starting_dag1` should have 5 neighbors. How many neighbors do `starting_dag2` and `starting_dag3` have?

(b) Write a function

`search_mec(starting_dag, ending_dag)`

which takes two Markov equivalent DAGs $\mathcal{G}$ and $\mathcal{H}$ and returns a sequence of covered edge reversals from $\mathcal{G}$ to $\mathcal{H}$.

The sequence of covered edges can be found by breadth-first search using `covered_edge_neighbors`. For breadth-first search, you might need to maintain a set of already-visited states. In Python, I recommend using a `frozenset` containing the edges of a DAG to represent each state, which is hashable and hence can be used in a set or as keys of a dictionary.

The shortest path from `starting_dag1` to `ending_dag1` should be of length 6 (where the length of the path is the number of covered edge flips). How long is the shortest path from `starting_dag2` to `ending_dag2`? How long is the shortest path from `starting_dag3` to `ending_dag3`?