CS 476/676 (Spring 2021): Homework 3

Due: Mar 3, 2021 at 11:59pm EST

Name: Joshua Popp

Instructions: This homework requires answering some open-ended questions, short proofs, and programming. This is an individual assignment, not group work. Though you may discuss the problems with your classmates, you must solve the problems and write the solutions independently. As stated in the syllabus, copying code from a classmate or the internet (even with minor changes) constitutes plagiarism. You are required to submit your answers in pdf form (use LaTeX) in a file called `<your-JHED>-hw3.pdf` to Gradescope under "HW3". Code should be submitted also to Gradescope under "HW3 Programming". Note that the autograder setup is intended only to register submission of the code, and will not provide any feedback. Code grading will be performed manually. Late submissions will be penalized, except in extenuating circumstances such as medical or family emergency. Submissions submitted 0-24 hours late will be penalized 10%, 24-48 hours late by 20%, 48-72 hours late by 30%, and later than 72 hours by 100%. Late days may be used (if available) to avoid these penalties.

The total assignment is worth 80 points + up to 6 points of extra credit.

Important note: For drawing graphs on this assignment, we encourage you to use the LaTeX template provided to draw graphs using the `tikz` package. However, hand drawing graphs/using other software and then including the images into the pdf using `includegraphics` is also acceptable as long as it is neat.

Important note: For the non-programming portions of the assignment that require drawing graphs, we encourage you to use the LaTeX template provided to draw graphs using the `tikz` package. However, hand drawing graphs/using other software and then including the images into the pdf using `includegraphics` is also acceptable as long as it is neat. For the programming portion, you **must** use a software package to produce a graphical visualization of the output of your code.

As an example of such external software that may produce acceptable results, you may consider using the `Ananke` Python package to draw undirected graphs, chain graphs and directed acyclic graphs. We have provided an example of its usage in the `hw3.py` code skeleton.

`https://ananke.readthedocs.io/en/latest/index.html` for installation and overview.
`https://ananke.readthedocs.io/en/latest/notebooks/causal_graphs.html` for how to draw various classes of graphs.

Problem 1 (6 points)

Consider an undirected graph $\mathcal{G}$ on defined over d variables $V_1, \ldots, V_d$. The clique factorization of UGs states that each clique potential must be **non-negative**. We claim that the odds ratio factorization of UGs is equivalent to the clique factorization. This must mean that the 1-way, 2-way, 3-way, $\ldots$, d-way terms that appear in the odds ratio factorization satisfy the non-negativity constraint. Prove that this is true. A simple inductive argument for higher order terms (greater than 2-way interactions) will suffice.

Taking a reference value of 0 for each $V_1, V_2, ..., V_d \in V$ as in class, our odds ratio factorization of $p(V)$ is as follows

$$p(V) = \frac{1}{Z} \times \prod_{V_i \in V} p\big(V_i \mid \text{nb}_{\mathcal{G}}(V_i) = 0\big) \times \prod_{\{V_i, V_j\} \in V} \text{OR}\left(V_i, V_j \mid \text{nb}_{\mathcal{G}}(V_i, V_j) = 0\right) \times \text{higher order terms}$$

Our normalization constant Z is non-negative by design, this is simply a positive factor taken to ensure our distribution is a valid probability distribution. Each first-order term is in itself a probability distribution, so $p\big(V_i \mid \text{nb}_{\mathcal{G}}(V_i) = 0\big) \in [0, 1] \implies p\big(V_i \mid \text{nb}_{\mathcal{G}}(V_i) = 0\big) \geq 0 \; \forall V_i \in V$. Second order terms consist of the odds ratio,

$$\text{OR}\left(V_i, V_j \mid \text{nb}_{\mathcal{G}}(V_i, V_j) = 0\right)$$

which we can also break down into the product of first-order probabilities using the definition of the odds ratio

$$\text{OR}\left(V_i, V_j \mid \text{nb}_{\mathcal{G}}(V_i, V_j) = 0\right) = \frac{p(V_i = x \mid V_j = y, \text{nb}_{\mathcal{G}}(V_i, V_j) = 0)}{p(V_i = 0 \mid V_j = y, \text{nb}_{\mathcal{G}}(V_i, V_j) = 0)} \times \frac{p(V_i = 0 \mid V_j = 0, \text{nb}_{\mathcal{G}}(V_i, V_j) = 0)}{p(V_i = x \mid V_j = 0, \text{nb}_{\mathcal{G}}(V_i, V_j) = 0)}$$

Note that in this example, we take reference values of 0. We cannot obtain negative numbers through the multiplication/ division of non-negative numbers, and this demonstrates that any odds ratio can be expressed as the product/ division of first-order probabilities, which are non-negative. Any higher order terms can be expressed by multiplying/ dividing pairwise odds ratios as demonstrated in class, so by induction these too can be decomposed into the product/ division of non-negative numbers, and consequently cannot break the non-negativity constraint.

Problem 2 (13 points)

Refer to Figure 1 for this problem.

1) Briefly describe the graphical property that the UG in Figure 1(a) satisfies, which allows it to be represented by a Markov equivalent DAG. (3 points)

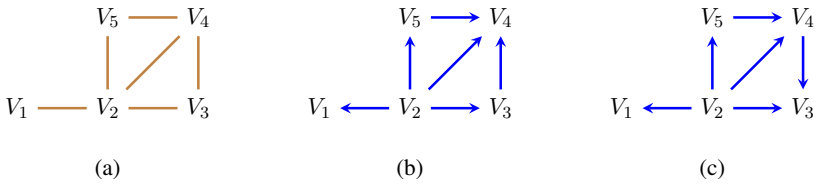


Figure 1

This is a chordal graph, meaning that any cycles of length 4 or greater (in this case, just $V_2 - V_3 - V_4 - V_5$) contains a chord

2) One of the DAGs in Figure 1 (b) or (c) is **not** Markov equivalent to the UG in (a) – prove which one by demonstrating a separation that holds in (a) via u-separation but not in (c) via d-separation. (5 points)

(b) is not Markov equivalent to the UG as $V_3 \perp\!\!\!\perp V_5 \mid V_4$ in (a) but not in (b) due to a collider

3) On careful examination, you will notice that the DAG chosen as the answer to part 2) has no Markov equivalent UG representation. What is the graphical property of this DAG that prevents this? Confirm that the other DAG does not have this issue. (5 points)

The DAG in (b) has an unshielded collider $V_3 \rightarrow V_4 \leftarrow V_5$ (meaning there exists no direct edge $V_3 \rightarrow V_5$ nor $V_3 \leftarrow V_5$).(c) does not have this issue. There are colliders at V_3 ($V_2 \rightarrow V_3 \leftarrow V_4$) and V_4 ($V_2 \rightarrow V_4 \leftarrow V_5$) but both of these have an edge between the co-parents ($V_2 \rightarrow V_4$ and $V_2 \rightarrow V_5$) so they are shielded.

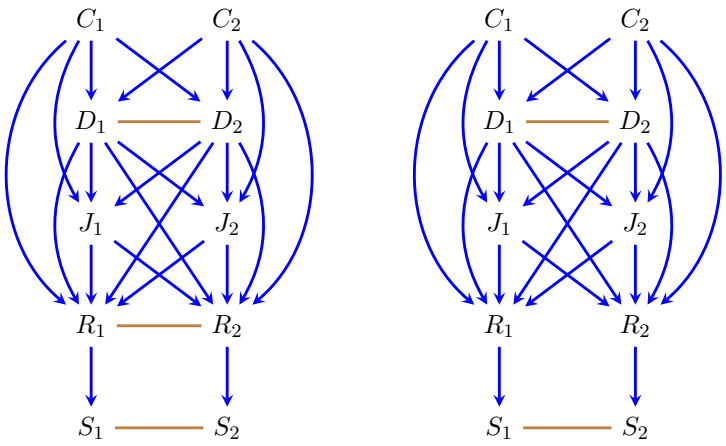
Problem 3 (15 points)

Refer to the chain graph model we used for our case study on the link between incarceration and engaging in risky sexual behaviors. The CG $\mathcal{G}$ is shown in Figure 2(a). Recall from our case study that R_1 and R_2 are binary variables encoding whether an individual engages in risky sexual behavior or not.

1) Briefly explain how $R_1 - R_2$ corresponds to social contagion. (3 points)

$R_1 - R_2$ refers to a direct influence between two individuals on the decision to engage in risky sexual behavior. This influence is an example of social contagion, as the decision on *whether to engage* in sexual behavior is, hopefully, not physical or biological in nature, but more a result of a social/ psychological/ sociological influence, such as peer pressure or joint decision-making by those involved.

2) Write down the odds ratio factorization up to 2-way terms for the piece $p(R_1, R_2 \mid \text{pa}_{\mathcal{G}}(R_1, R_2))$ that appears in the first-level of the CG factorization for $\mathcal{G}$. Use a reference value of 0. (5 points)



(a) $\mathcal{G}$ (b) $\mathcal{G}$ under the null

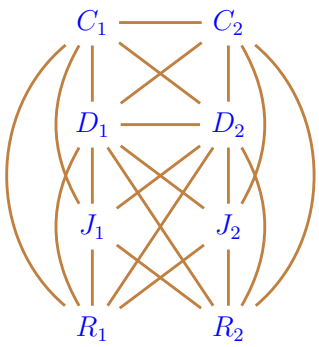
Figure 2

$$
\begin{aligned}
p(R_1, R_2 \mid \mathrm{pa}_{\mathcal{G}}(R_1, R_2)) &= \frac{1}{Z(\mathrm{pa}_{\mathcal{G}}(R_1, R_2))} \times p\big(R_1 \mid \mathrm{nb}_{\mathcal{G}}(R_1) = 0, \mathrm{pa}_{\mathcal{G}}(R_1)\big) \\
&\quad \times p\big(R_2 \mid \mathrm{nb}_{\mathcal{G}}(R_2) = 0, \mathrm{pa}_{\mathcal{G}}(R_2)\big) \\
&\quad \times \textbf{OR}\left(R_1, R_2 \mid \mathrm{nb}_{\mathcal{G}}(R_1, R_2) = 0, \mathrm{pa}_{\mathcal{G}}(R_1, R_2)\right) \\
&= \frac{1}{Z(\mathrm{pa}_{\mathcal{G}}(R_1, R_2))} \times p\big(R_1 \mid R_2 = 0, \mathrm{pa}_{\mathcal{G}}(R_1)\big) \\
&\quad \times p\big(R_2 \mid R_1 = 0, \mathrm{pa}_{\mathcal{G}}(R_2)\big) \\
&\quad \times \textbf{OR}\left(R_1, R_2 \mid \mathrm{pa}_{\mathcal{G}}(R_1, R_2)\right)
\end{aligned}
$$

where

$$
\begin{aligned}
\mathrm{pa}_{\mathcal{G}}(R_1) &= C_1, D_1, D_2, J_1, J_2 \\
\mathrm{pa}_{\mathcal{G}}(R_2) &= C_2, D_1, D_2, J_1, J_2 \\
\mathrm{pa}_{\mathcal{G}}(R_1, R_2) &= C_1, C_2, D_1, D_2, J_1, J_2
\end{aligned}
$$

3) The CG under the null hypothesis of no social contagion via $R_1 - R_2$ is shown in Figure 2(b). Show (using c-separation – including drawing the UG corresponding to the augmented graph) that a term appearing in the odds ratio factorization you wrote down in part 1) is a valid measure of causal association for testing this causal null hypothesis. (7 points)



$(\mathcal{G}_{\text{pan}_\mathcal{G}(R_1 \cup R_2)})^a$

In this augmented graph $(\mathcal{G}_{\text{pan}_\mathcal{G}(R_1 \cup R_2)})^a$, $R_1 \perp\!\!\!\perp_{\text{u-sep}} R_2 \mid \{C_1, C_2, D_1, D_2, J_1, J_2\}$. This implies that $R_1 \perp\!\!\!\perp_{\text{c-sep}} R_2 \mid \text{pa}_\mathcal{G}(R_1, R_2)$ in $\mathcal{G}$ since $\text{pa}_\mathcal{G}(R_1, R_2) = \{C_1, C_2, D_1, D_2, J_1, J_2\}$. Assuming faithfulness to $\mathcal{G}$ it follows that the term $\text{OR}(R_1, R_2 \mid \text{pa}_\mathcal{G}(R_1, R_2))$ from our factorization above is a valid measure of causal association for this test of the null of no social contagion.

Problem 4 (45 points + up to 6 points of extra credit)

This problem contains an analysis component and a programming component. The objective is to help you understand how to move from a novel theoretical result to implementing this idea to gain insights from data.

1) A directed (but not necessarily acyclic) graph $\mathcal{G}$ on d variables $V_1, \ldots, V_d$ can be summarized as a $d \times d$ binary adjacency matrix A as follows: $A_{ij} = 1$ if $V_i \to V_j$ exists in $\mathcal{G}$, and $A_{ij} = 0$ otherwise.[1] Let e^A be defined as the matrix exponential, whose Taylor expansion is quite similar to the standard scalar exponential,

$$e^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k.$$

Prove that $\text{trace}(e^A) - d = 0$ **if and only if** $\mathcal{G}$ is acyclic, i.e., $\mathcal{G}$ is a DAG, where $\text{trace}(\cdot)$ is the sum of the diagonal elements of the matrix. (7 points)

Hint: you may use the fact that A^k_{ij} counts the number of directed walks of length k from a vertex V_i to V_j.

[1] A_{ij} denotes the (i, j)-th entry of the matrix A.

First, note that the following are all equivalent

$$\begin{aligned}
\text{trace}(e^A) - d &= 0 \\
\text{trace}(e^A) &= d \\
\text{trace}(\sum_{k=0}^{\infty} \frac{1}{k!} A^k) &= d \\
\sum_{k=0}^{\infty} \sum_{i=1}^{d} \frac{1}{k!} A_{ii}^k &= d
\end{aligned}$$

First, we show that **if** A is acyclic **then** $\text{trace}(e^A) - d = 0$. Note that if A is acyclic then there exists one and only one path from $V_i \to V_i$, a path of length zero. Using the hint, we see that

$$\begin{aligned}
\sum_{k=0}^{\infty} \sum_{i=1}^{d} \frac{1}{k!} A_{ii}^k &= \sum_{i=1}^{d} \frac{1}{0!} A_{ii}^0 \\
&= \sum_{i=1}^{d} 1 \\
&= d \text{ (QED)}
\end{aligned}$$

Next, we show that **if** A is **not** acyclic **then** $\text{trace}(e^A) - d \neq 0$. Assume A has a cycle of length l from $V_i \to ... \to V_i$. Then for $k = l$, there is at least one path of length k from V_i to V_i. Noting also that all zero-length paths from V_i to self remain and $\frac{1}{k!} A_{ii}^k \geq 0 \; \forall i, k$, we have

$$\begin{aligned}
\sum_{k=0}^{\infty} \sum_{i=1}^{d} \frac{1}{k!} A_{ii}^k &= \sum_{i=1}^{d} \frac{1}{0!} A_{ii}^0 + \sum_{k=1}^{\infty} \sum_{i=1}^{d} \frac{1}{k!} A_{ii}^k \\
&= \sum_{i=1}^{d} \frac{1}{0!} A_{ii}^0 + \sum_{i=1}^{d} \frac{1}{l!} A_{ii}^l + ... \\
&> \sum_{i=1}^{d} \frac{1}{0!} A_{ii}^0 = d
\end{aligned}$$

So for any cyclic graph, $\text{trace}(e^A) > d$

2) Implement the above constraint as a Python function in the `hw3.py` skeleton. The function takes as input a binary adjacency matrix in the form of a `numpy` array. Run the provided test cases to confirm that your implemented constraint is correct. (3 points)

3) For the rest of Problem 4 we will assume that the data are generated from a DAG with a linear Gaussian SEM (Gaussians belong to the curved exponential family) as follows,

$$V_i \leftarrow \beta_0 + \sum_{V_j \in \text{pa}_{\mathcal{G}}(V_i)} \beta_j V_j + \epsilon_i,$$

where noise terms ϵ_i are drawn from a normal distribution and are mutually independent of each other. That is, each variable is generated as a linear function of an intercept term, its parents, and a Gaussian noise term.

We saw in lecture that the Bayesian information criterion (BIC) is a **consistent** score for model selection when the underlying data generating process comes from the curved exponential family. Similar to how the joint distribution for a DAG factorizes as a product of conditionals of each variable given its parents – $p(V_i \mid \text{pa}_{\mathcal{G}}(V_i))$ – the BIC score for DAGs also decomposes as the sum of BIC scores for individual models fit for each variable given its parents. That is, for DAGs,

$$\text{BIC} = \sum_{V_i \in V} \text{BIC}_{V_i|\text{pa}_{\mathcal{G}}(V_i)}$$

Implement the linear Gaussian BIC score for DAGs as a Python function in the `hw3.py` skeleton as follows. The function accepts an adjacency matrix encoding a DAG, a `pandas` data frame containing the data, and a Python dictionary mapping row/column indices in the adjacency matrix to variable names in the data frame. For each variable V_i, use the structure of the adjacency matrix to fit a linear regression of V_i as a function of an intercept term and its parents using ordinary least squares from the `ols` module of the `statsmodels` package. Then, $\text{BIC}_{V_i|\text{pa}_{\mathcal{G}}(V_i)}$ can be directly obtained as an attribute from this fitted model.[2] Use the decomposability of the BIC for DAGs as mentioned above to compute the BIC for the entire model.
(7 points)

4) Run your implemented BIC function on the test cases provided in the `hw3.py` skeleton and answer the following questions based on the tests. Please provide the computed BICs and answers to the questions **explicitly** in the PDF submission.

a) Based on their BICs, which of $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_4$ would you consider to be Markov equivalent? (3 points)

$\mathcal{G}_1, \mathcal{G}_2$, and $\mathcal{G}_4$ each have a BIC of 23030.573, while $\mathcal{G}_3$ has a BIC of 22548.647

This suggests that $\mathcal{G}_1, \mathcal{G}_2$, and $\mathcal{G}_4$ are Markov equivalent, and $\mathcal{G}_3$ is not Markov equivalent to the rest.

b) If the true data generating process is a linear Gaussian SEM corresponding to one of the 4 DAGs, which of the models would the BIC suggest picking as the true underlying model? (3 points)

The BIC decreases as log likelihood increases, and increases as complexity increases, so $\mathcal{G}_3$, which has the lowest BIC, is the best pick for the true underlying model.

5) Similar to HW2, you are provided data that is drawn from the paper by Sachs et al. (2005) in the file `data.txt`. Your goal for this homework is to implement a "causal discovery" or "structure learning" procedure where you will start with an empty DAG (a DAG with no edges) over all of the proteins, and try to find one which fits the data best according to the BIC function you have just implemented. Your implementation of the acyclicity constraint in part 1) will ensure your procedure rejects any graphs that are not acyclic. Ultimately, you will qualitatively compare the

[2] If you're curious about the explicit calculations for the BIC from linear regressions, check out the Gaussian special case of the BIC here: `https://en.wikipedia.org/wiki/Bayesian_information_criterion`.

DAG learned by your procedure to the one inferred by Sachs et al. (2005) in the original paper.

A simple procedure to find the DAG with the optimal BIC is to simply enumerate all possible DAGs on the variables provided and pick the ones that have the lowest BIC score. The given dataset contains 10 proteins. The number of possible DAGs on d variables grows super-exponentially – so, on 10 variables we have 4175098976430598143 possible DAGs! So enumerating all DAGs and scoring them is clearly impossible. Instead, we will implement a greedy approach to causal discovery as follows.

Implement the greedy causal discovery algorithm in Algorithm 1 (last page of this PDF) using the skeleton provided in `hw3.py`. (16 points)

Helpful tip: set the number of steps to be low (maybe 10 or so) as you debug your implementation.

6) Visualize the final DAG learned by running your implementation for 100 steps. Include an image of the graph in your PDF submission. We have provided you code to do the visualization using the `Ananke` package in the skeleton `hw3.py`. If you wish you to use a different package you may do so. Examine the final DAG and write a few sentences about how it compares to the DAG learned by Sachs et al. (2005) shown in Figure 3. E.g., is the Raf $\rightarrow$ Mek $\rightarrow$ Erk cascade present in your DAG? Provide any other comparisons you find interesting. We are not evaluating you on how well your DAG compares to the original one – it turns out several researchers have tried to recreate it and failed. We just want you to get practice visualizing and interpreting outputs from your code in preparation for the final project. (6 points)

Our DAG appears to be pretty different from that learned by Sachs et al. The Raf $\rightarrow$ Mek $\rightarrow$ Erk cascade is not present in our DAG, we find Mek to be a parent of both Erk and Raf. A notable difference is that our DAG is much more dense than the one Sachs et al. learned, and quite a few of the edges that are shared have reversed orientation (such as aforementioned Raf $\rightarrow$ Mek, as well as Erk $\rightarrow$ Akt). However, one interesting point is that we do also see that PKC and PKA have quite a lot of edges. These types of hubs can be very important in biological networks like this one, so it's worth noting whether some of the same hubs appear, and this would be one potential area for further focus (this association isn't too strong, as Erk for example also has many edges in our graph but not in theirs). This preservation of hubs could be a point to come back to if we were to look at alternative structure learning techniques, especially any that more strictly enforce sparsity.

Extra credit: Make the implementation better in any one of the following ways:

1. Design a simple new heuristic or move and justify why it might help in addition to the ones already present. Feel free to get really creative here! (3 points)
2. We know the BIC is decomposable, which means that each edge addition, deletion, or reversal does not require recomputing the BIC for every variable – only the ones whose parent sets changed. Improve computational efficiency of the algorithm by taking advantage of the BIC's decomposability. Note: you might feel your current procedure is slow, but parallel implementations that take advantage of the decomposability of the BIC have been run on problems with millions of variables! (6 points)
3. We know the BIC for Markov equivalent graphs are roughly equal. This can be used to keep

track of any DAGs we find along the way that are Markov equivalent to our final DAG, as a way of reporting uncertainty in the edge orientations (we will learn more about this in a week or two.) Implement a separate function CAUSAL DISCOVERY EQUIVALENCE that augments your causal discovery function to also collect Markov equivalent DAGs in the final solution. We are asking for a separate function here so that it does not mess up our autograder. (5 points)

I implemented number 2 in the function CAUSAL DISCOVERY EFFICIENT. Since the BIC is decomposable into a sum of terms from each node based solely on that node's parents, we can improve efficiency by keeping track of $\text{BIC}^*_{\text{nodewise}}$, a list of d terms, one for each node, such that $\text{BIC}^*_{\text{nodewise}}[i] = \text{BIC}_{V_i|\text{pa}_\mathcal{G}(V_i)}$. Then for each edge manipulation, we only have to recompute the BIC for the nodes whose parents changed (the tail for edge addition/deletion, both head and tail for reversal) rather than the full graph. This procedure learns the exact same DAG, but runtime improved from 23.1 seconds to 4.165 seconds for 100 steps. This application of the decomposed BIC would scale much better to larger graphs.

You can get a max of up to 6 points of extra credit for implementing any one of these (you will not get extra points for implementing multiple bonus features but you can do so if you wish.)

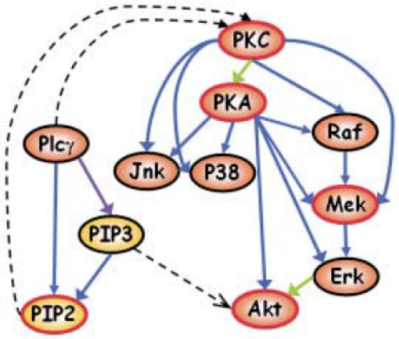
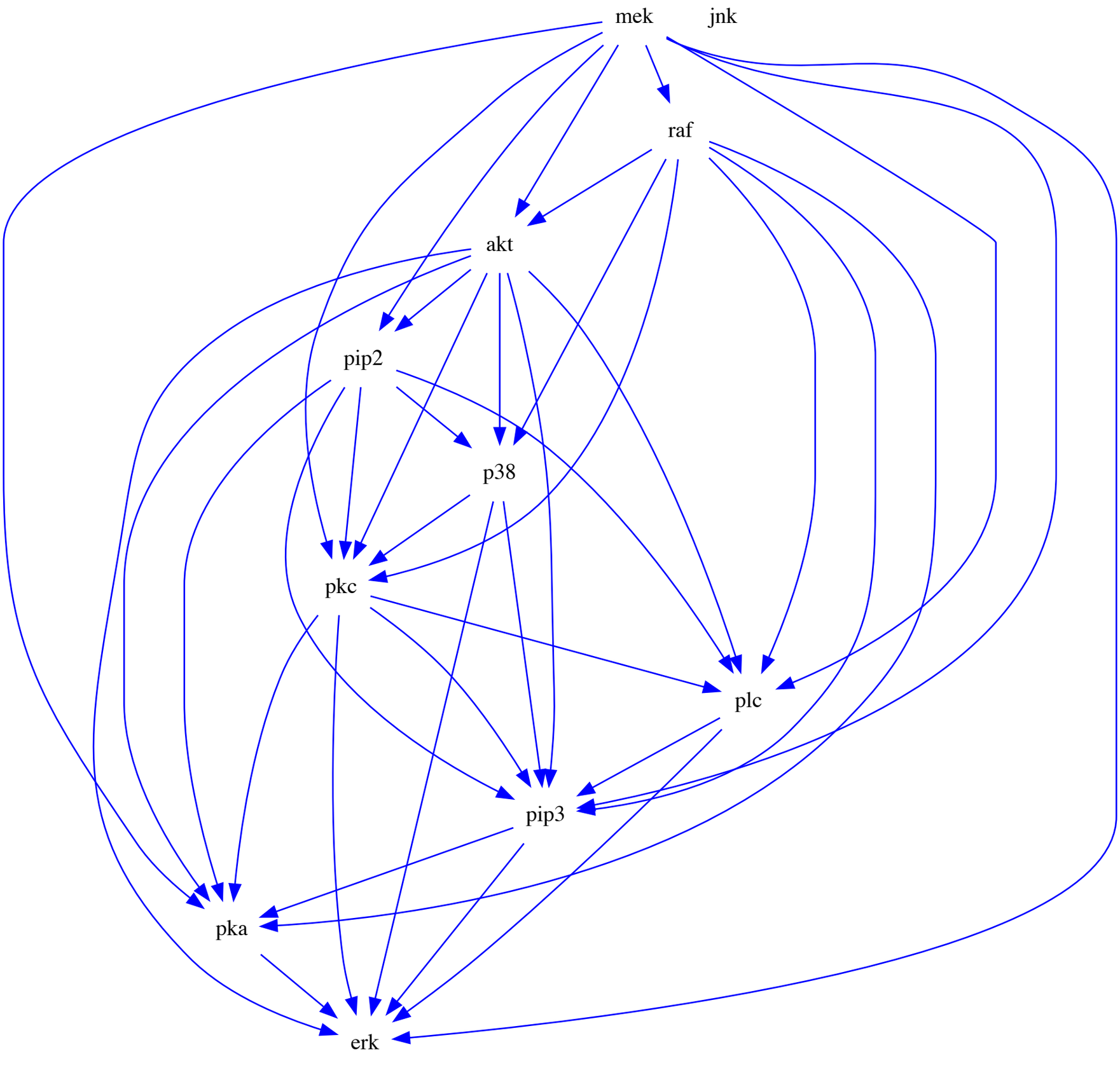


Figure 3: The DAG learned from phospho-protein/phospho-lipid expression data by Sachs et al. (2005)

References

Sachs, K., Perez, O., Pe'er, D., Lauffenburger, D. A., and Nolan, G. P. (2005). Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308(5721):523–529.


mek
jnk
raf
akt
pip2
p38
pkc
plc
pip3
pka
erk

Algorithm 1 CAUSAL DISCOVERY

Inputs: $\mathcal{D}$ pandas data frame, num_steps (default 100), ϵ cycle score tolerance (default $1e^{-9}$)

Initialize empty DAG as A^* a $d \times d$ matrix with all zeros

Initialize an empty set E to store edges in the graph for easy deletion/reversal moves

Set up $\mathcal{M}$ as a map from row/column indices in A^* to variable names in $\mathcal{D}$

Initialize BIC* as BIC$(A^*, \mathcal{D}, \mathcal{M})$ ▷ using the function implemented in part 3)

for i in $(1, \ldots,$ num_steps$)$ **do**

There are 3 types of moves we will consider at each step – random edge **addition**, random edge **deletion**, and random edge **reversal**. We will score the BIC of each move and pick the best if it is better than the current DAG in A^*

For random edge addition:

- Generate random indices i, j
- Check setting A^*_{ij} to 1 yielding a new matrix A^*_{add} is still a DAG, i.e., CYCLE SCORE$(A^*_{\text{add}}) < \epsilon$
- If it is a DAG, store BIC$_{\text{add}}$ as BIC$(A^*_{\text{add}}, \mathcal{D}, \mathcal{M})$

For random edge deletion:

- Choose a random edge from the set of existing edges E
- Set A^*_{ij} to 0 yielding a new matrix A^*_{delete} (this is guaranteed to be a DAG)
- Store BIC$_{\text{delete}}$ as BIC$(A^*_{\text{delete}}, \mathcal{D}, \mathcal{M})$

For random edge reversal:

- Choose a random edge from the set of existing edges E
- Set A^*_{ij} to 0 and A^*_{ji} to 1 yielding a new matrix A^*_{rev} and apply the acyclicity check to A^*_{rev}.
- If it is a DAG, store BIC$_{\text{rev}}$ as BIC$(A^*_{\text{rev}}, \mathcal{D}, \mathcal{M})$

Compare BIC$_{\text{add}}$, BIC$_{\text{delete}}$, and BIC$_{\text{rev}}$ and pick the lowest (best.) Call it BIC$_{\text{move}}$

If BIC$_{\text{move}}$ < BIC* then apply the move (add, delete, reverse as appropriate) to A^* and the set of edges E and update BIC* to BIC$_{\text{move}}$

return $A^*, E, \mathcal{M}$