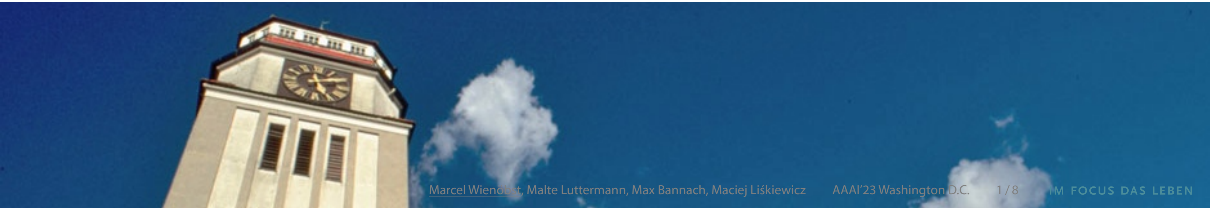
UNIVERSITÄT ZU LÜBECK

Efficient Enumeration of Markov Equivalent DAGs

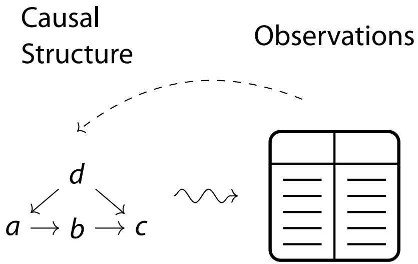
Marcel Wienöbst Malte Luttermann Max Bannach Maciej Liśkiewicz

AAAI'23 Washington D.C.

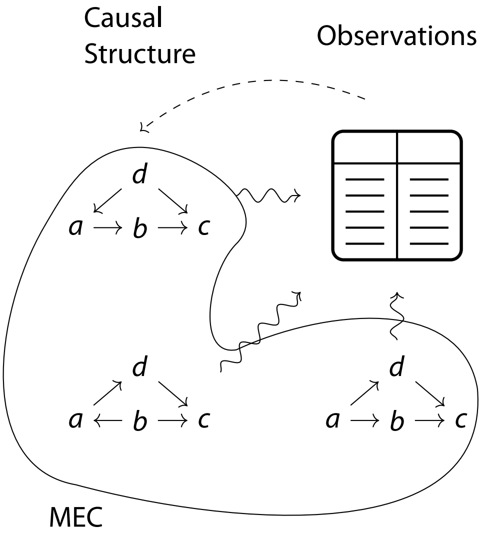
IM FOCUS DAS LEBEN



Motivation and Problem Setup

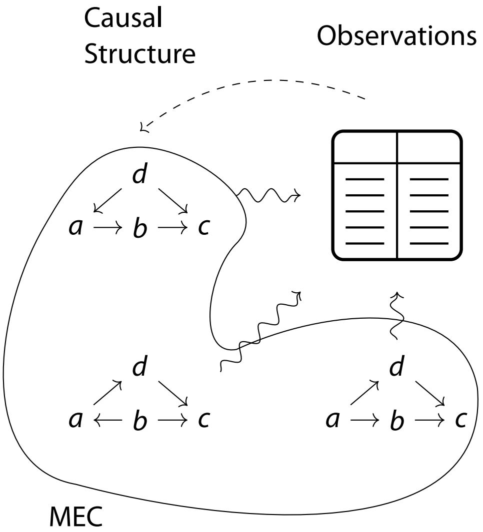


Motivation and Problem Setup



Without further assumptions, the causal structure can only be recovered up to Markov equivalence.

Motivation and Problem Setup



Without further assumptions, the causal structure can only be recovered up to Markov equivalence.

Problem

Enumerate all directed acyclic graphs (DAGs) in a Markov equivalence class (MEC) efficiently.

Efficient algorithm to enumerate all member DAGs of a Markov equivalence class

Asked 8 years, 4 months ago Modified 8 years, 4 months ago Viewed 550 times

4

I'm working on a research project involving Bayesian networks. BNs are directed acyclic graphs (DAGs) used to compactly represent joint distributions of variables. In many cases, multiple DAGs can represent the same distribution, and so compose an equivalence class. Equivalent DAGs share the same skeleton (same edges if one ignores direction), and the orientation of edges in their v-structures (X->Y, Z->Y, but no edge between X and Z) must be the same. Such equivalence classes can be represented by partially directed acyclic graphs (PDAGs), with the *compelled* (i.e. have the same orientation) edges directed and the *reversible* edges undirected.

Given a DAG, I would like to be able to find all other DAGs in its equivalence class. I'm aware of an efficient algorithm (Chickering 1995) that, given a DAG, finds the complete PDAG representing its equivalence class. I'm also aware of an algorithm (Dor and Tarsi 1992) that, given a PDAG, generates a random member DAG of the equivalence class. I am not interested in a random member, but rather in enumerating all members.

It might seem trivial at first glance--why not just try all combinations of orientations of the reversible edges, and discard the ones that aren't acyclic? But the number of possibilities grows exponentially in the number of reversible edges, so I'm worried that this won't work for large graphs. I've seen some sources claim that the proportion of edges that are reversible is relatively small, but still, if the graph is large enough, it will still be problematic. I'm working with graphs of up to several hundred variables, with several thousand being a possibility later on.

algorithms graphical-model graph-theory bayesian-network

Share Cite Improve this question Follow

asked Oct 10, 2014 at 18:57
jdj
41 1

Feature
Acc

Relatec

5 Im
3 Ma m
2 Wl ne in
0 Ho in Ma

Hot Ne

At wha circuits
A SF hu a plug bar of s
Accord denies transge
Functic functio
Why dc grains?
Applyir

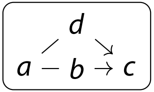
Is there some way you could avoid doing this? For example, Maathuis et al. (2009) started with a problem that looked like it would require enumerating all of the DAGs, and showed that it could be solved by finding all possible *parent sets of each node*. They were also motivated by a high-dimensional problem (5,000 nodes). What is the problem you are trying to solve using the list of DAGs? – Lizzie Silver Oct 16, 2014 at 15:26

1 A way to do this is the following: pick any undirected edge, try both orientations and recurse. After applying an orientation of an edge, apply Meek's rules to propagate changes (arxiv.org/ftp/arxiv/papers/1302/1302.4972.pdf). – George Oct 19, 2014 at 22:50

Add a comment

Formal Problem and its History

CPDAG



Input: Representation of an MEC as CPDAG.

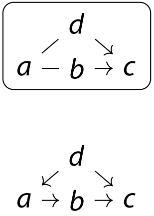
Task: List all DAGs in MEC one-by-one.

Objective: Small *delay* between successive outputs.

	Approach	Delay
Meek '95	Meek-Rule Recursion	$O(m \cdot \text{meek}(n, m))$
Chickering '95	Transformational Characterization	$O(m^3)$
This work	Max. Cardinality Search (MCS)	$O(n + m)$

Formal Problem and its History

CPDAG



Input: Representation of an MEC as CPDAG.

Task: List all DAGs in MEC one-by-one.

Objective: Small *delay* between successive outputs.

	Approach	Delay
Meek '95	Meek-Rule Recursion	$O(m \cdot \text{meek}(n, m))$
Chickering '95	Transformational Characterization	$O(m^3)$
This work	Max. Cardinality Search (MCS)	$O(n + m)$

Formal Problem and its History

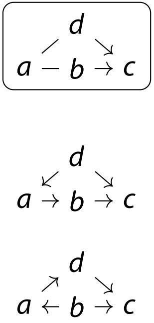
Input: Representation of an MEC as CPDAG.

Task: List all DAGs in MEC one-by-one.

Objective: Small *delay* between successive outputs.

	Approach	Delay
Meek '95	Meek-Rule Recursion	$O(m \cdot \text{meek}(n, m))$
Chickering '95	Transformational Characterization	$O(m^3)$
This work	Max. Cardinality Search (MCS)	$O(n + m)$

CPDAG



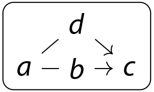
Formal Problem and its History

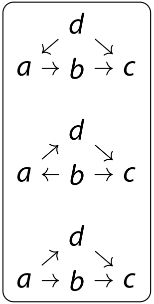
Input: Representation of an MEC as CPDAG.

Task: List all DAGs in MEC one-by-one.

Objective: Small *delay* between successive outputs.

CPDAG





MEC

	Approach	Delay
Meek '95	Meek-Rule Recursion	$O(m \cdot \text{meek}(n, m))$
Chickering '95	Transformational Characterization	$O(m^3)$
This work	Max. Cardinality Search (MCS)	$O(n + m)$

Prerequisite: Extending a CPDAG

Input: Representation of an MEC as CPDAG.

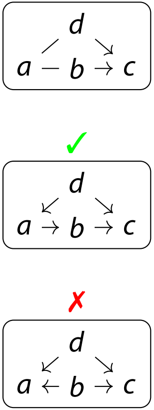
Task: Compute any DAG in the MEC.

Algorithm: Folklore $O(n+m)$ approach.

(i) Discard all directed edges.

(ii) Find acyclic orientation without v-structure using MCS (traverse vertices by highest number of visited neighbors).

CPDAG



Prerequisite: Extending a CPDAG

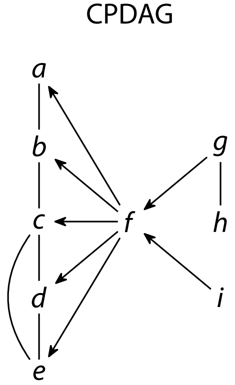
Input: Representation of an MEC as CPDAG.

Task: Compute any DAG in the MEC.

Algorithm: Folklore $O(n + m)$ approach.

(i) Discard all directed edges.

(ii) Find acyclic orientation without v-structure using MCS (traverse vertices by highest number of visited neighbors).



Prerequisite: Extending a CPDAG

Input: Representation of an MEC as CPDAG.

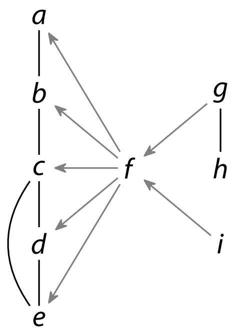
Task: Compute any DAG in the MEC.

Algorithm: Folklore $O(n+m)$ approach.

(i) Discard all directed edges.

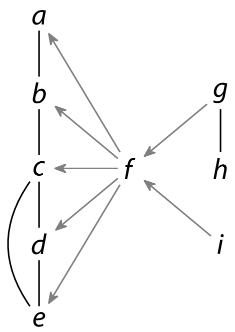
(ii) Find acyclic orientation without v-structure using MCS (traverse vertices by highest number of visited neighbors).

Step (i)



Prerequisite: Extending a CPDAG

Step (ii)



Input: Representation of an MEC as CPDAG.

Task: Compute any DAG in the MEC.

Algorithm: Folklore $O(n+m)$ approach.

(i) Discard all directed edges.

(ii) Find acyclic orientation without v-structure using MCS (traverse vertices by highest number of visited neighbors).

Number of visited neighbors:

a	b	c	d	e	f	g	h	i
0	0	0	0	0	0	0	0	0

Prerequisite: Extending a CPDAG

Input: Representation of an MEC as CPDAG.

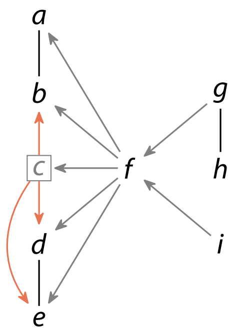
Task: Compute any DAG in the MEC.

Algorithm: Folklore $O(n + m)$ approach.

(i) Discard all directed edges.

(ii) Find acyclic orientation without v-structure using MCS (traverse vertices by highest number of visited neighbors).

Step (ii)



Number of visited neighbors:

a	b	c	d	e	f	g	h	i
0	1	0	1	1	0	0	0	0

Prerequisite: Extending a CPDAG

Step (ii)

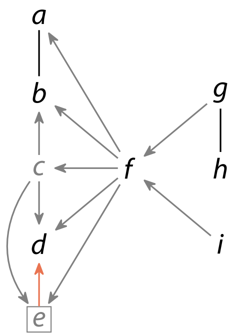
Input: Representation of an MEC as CPDAG.

Task: Compute any DAG in the MEC.

Algorithm: Folklore $O(n+m)$ approach.

(i) Discard all directed edges.

(ii) Find acyclic orientation without v-structure using MCS (traverse vertices by highest number of visited neighbors).

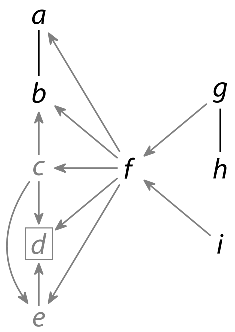


Number of visited neighbors:

a	b	c	d	e	f	g	h	i
0	1	0	2	1	0	0	0	0

Prerequisite: Extending a CPDAG

Step (ii)



Input: Representation of an MEC as CPDAG.

Task: Compute any DAG in the MEC.

Algorithm: Folklore $O(n+m)$ approach.

(i) Discard all directed edges.

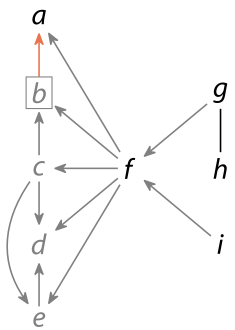
(ii) Find acyclic orientation without v-structure using MCS (traverse vertices by highest number of visited neighbors).

Number of visited neighbors:

a	b	c	d	e	f	g	h	i
0	1	0	2	1	0	0	0	0

Prerequisite: Extending a CPDAG

Step (ii)



Input: Representation of an MEC as CPDAG.

Task: Compute any DAG in the MEC.

Algorithm: Folklore $O(n+m)$ approach.

(i) Discard all directed edges.

(ii) Find acyclic orientation without v-structure using MCS (traverse vertices by highest number of visited neighbors).

Number of visited neighbors:

a	b	c	d	e	f	g	h	i
1	1	0	2	1	0	0	0	0

Prerequisite: Extending a CPDAG

Input: Representation of an MEC as CPDAG.

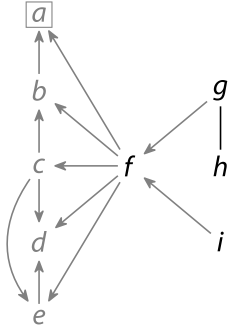
Task: Compute any DAG in the MEC.

Algorithm: Folklore $O(n + m)$ approach.

(i) Discard all directed edges.

(ii) Find acyclic orientation without v-structure using MCS (traverse vertices by highest number of visited neighbors).

Step (ii)



Number of visited neighbors:

a	b	c	d	e	f	g	h	i
1	1	0	2	1	0	0	0	0

Prerequisite: Extending a CPDAG

Input: Representation of an MEC as CPDAG.

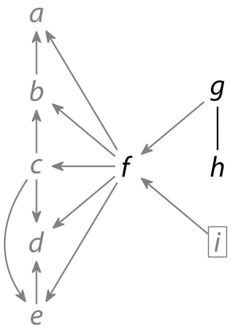
Task: Compute any DAG in the MEC.

Algorithm: Folklore $O(n+m)$ approach.

(i) Discard all directed edges.

(ii) Find acyclic orientation without v-structure using MCS (traverse vertices by highest number of visited neighbors).

Step (ii)



Number of visited neighbors:

a	b	c	d	e	f	g	h	i
1	1	0	2	1	0	0	0	0

Prerequisite: Extending a CPDAG

Input: Representation of an MEC as CPDAG.

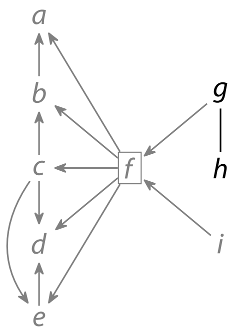
Task: Compute any DAG in the MEC.

Algorithm: Folklore $O(n+m)$ approach.

(i) Discard all directed edges.

(ii) Find acyclic orientation without v-structure using MCS (traverse vertices by highest number of visited neighbors).

Step (ii)



Number of visited neighbors:

a	b	c	d	e	f	g	h	i
1	1	0	2	1	0	0	0	0

Prerequisite: Extending a CPDAG

Input: Representation of an MEC as CPDAG.

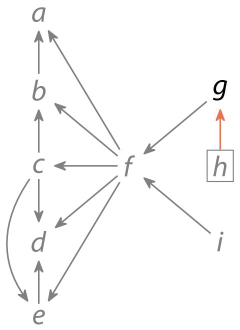
Task: Compute any DAG in the MEC.

Algorithm: Folklore $O(n + m)$ approach.

(i) Discard all directed edges.

(ii) Find acyclic orientation without v-structure using MCS (traverse vertices by highest number of visited neighbors).

Step (ii)



Number of visited neighbors:

a	b	c	d	e	f	g	h	i
1	1	0	2	1	0	1	0	0

Prerequisite: Extending a CPDAG

Input: Representation of an MEC as CPDAG.

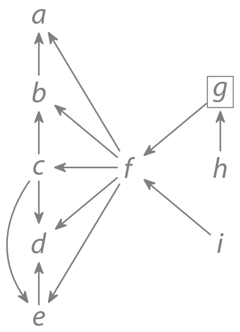
Task: Compute any DAG in the MEC.

Algorithm: Folklore $O(n + m)$ approach.

(i) Discard all directed edges.

(ii) Find acyclic orientation without v-structure using MCS (traverse vertices by highest number of visited neighbors).

Step (ii)



Number of visited neighbors:

a	b	c	d	e	f	g	h	i
1	1	0	2	1	0	1	0	0

Prerequisite: Extending a CPDAG

Input: Representation of an MEC as CPDAG.

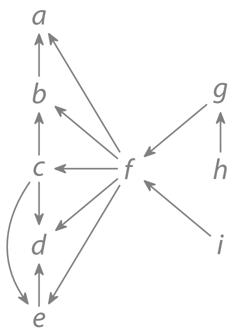
Task: Compute any DAG in the MEC.

Algorithm: Folklore $O(n + m)$ approach.

(i) Discard all directed edges.

(ii) Find acyclic orientation without v-structure using MCS (traverse vertices by highest number of visited neighbors).

Step (ii)

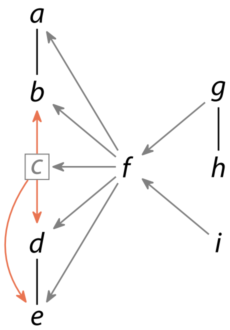


Number of visited neighbors:

a	b	c	d	e	f	g	h	i
1	1	0	2	1	0	1	0	0

Linear-Time Enumeration Algorithm

Visiting c first:

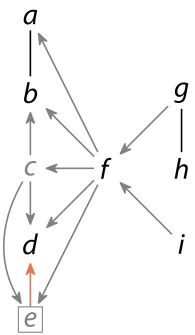


Number of visited neighbors:

a	b	c	d	e	f	g	h	i
0	1	0	1	1	0	0	0	0

Linear-Time Enumeration Algorithm

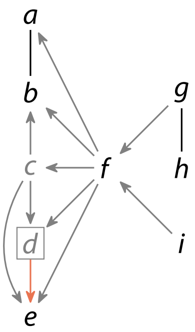
Visiting e after c:



Number of visited neighbors:

a	b	c	d	e	f	g	h	i
0	1	0	2	1	0	0	0	0

Visiting d after c:

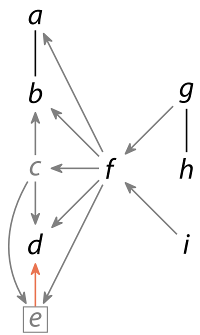


Number of visited neighbors:

a	b	c	d	e	f	g	h	i
0	1	0	1	2	0	0	0	0

Linear-Time Enumeration Algorithm

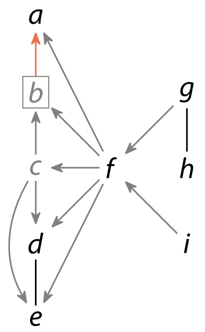
Visiting e after c:



Number of visited neighbors:

a	b	c	d	e	f	g	h	i
0	1	0	2	1	0	0	0	0

Visiting b after c:

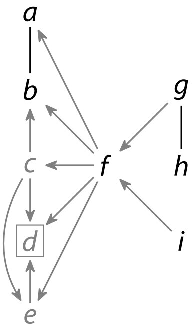


Number of visited neighbors:

a	b	c	d	e	f	g	h	i
1	1	0	1	1	0	0	0	0

Linear-Time Enumeration Algorithm

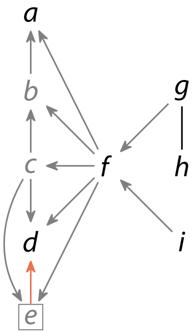
Visiting e after c:



Number of visited neighbors:

a	b	c	d	e	f	g	h	i
0	1	0	2	1	0	0	0	0

Visiting b after c:

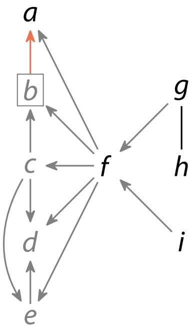


Number of visited neighbors:

a	b	c	d	e	f	g	h	i
1	1	0	2	1	0	0	0	0

Linear-Time Enumeration Algorithm

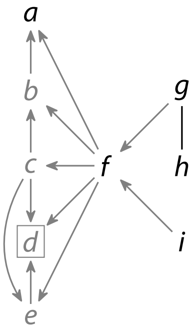
Visiting e after c:



Number of visited neighbors:

a	b	c	d	e	f	g	h	i
1	1	0	2	1	0	0	0	0

Visiting b after c:

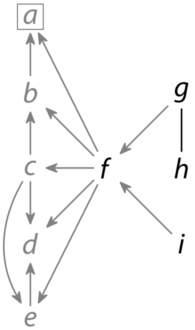


Number of visited neighbors:

a	b	c	d	e	f	g	h	i
1	1	0	2	1	0	0	0	0

Linear-Time Enumeration Algorithm

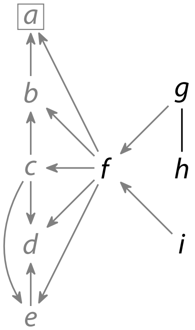
Visiting e after c:



Number of visited neighbors:

a	b	c	d	e	f	g	h	i
1	1	0	2	1	0	0	0	0

Visiting b after c:



Number of visited neighbors:

a	b	c	d	e	f	g	h	i
1	1	0	2	1	0	0	0	0

Linear-Time Enumeration Algorithm

Lemma

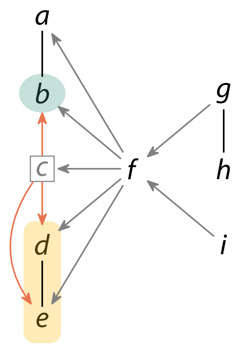
If vertices x and y are…

- *…connected: choosing either vertex first results in disjoint extensions.*
- *…unconnected: any extension produced by choosing x first, can also be produced by choosing a vertex from the connected component of y first.*

Theorem

- *An MEC can be enumerated with delay $O(n+m)$.*
- *For background knowledge, an $O(n^3)$ initialization step is needed, subsequent delay is $O(n+m)$.*

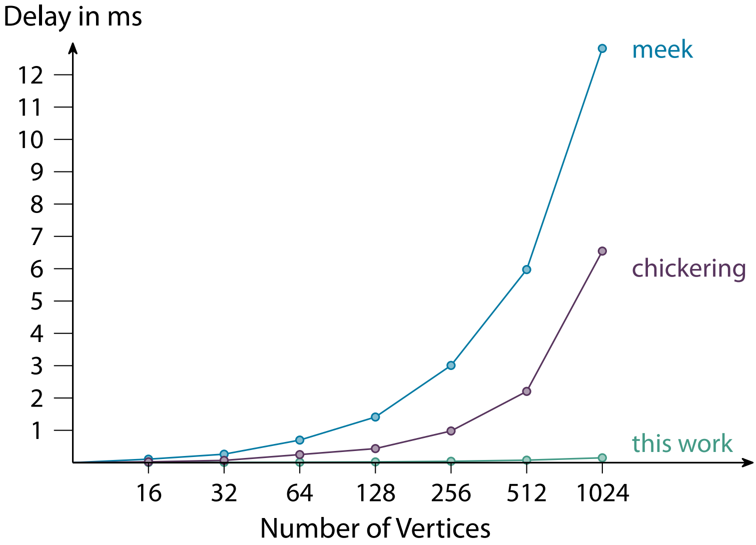
Visiting c first:



Number of visited neighbors:

a	b	c	d	e	f	g	h
0	1	0	1	1	0	0	0

Experimental Evaluation



Outlook

Theorem (Another structural result)

Every Markov equivalence class can be enumerated such that successive DAGs have structural hamming distance (SHD) ≤ 3.

Outlook

Theorem (Another structural result)

Every Markov equivalence class can be enumerated such that successive DAGs have structural hamming distance (SHD) ≤ 3.

Open Problem: Enumeration of Markov equivalent *MAGs* (causal models under latent confounding).

Outlook

Theorem (Another structural result)

Every Markov equivalence class can be enumerated such that successive DAGs have structural hamming distance (SHD) ≤ 3.

Open Problem: Enumeration of Markov equivalent *MAGs* (causal models under latent confounding).

Thanks for your attention!

Code and Preprint available on Github:
`github.com/mwien/fastmecenumeration`