MCMC: Metropolis-Hastings algorithm

Ingredient: A procedure "modify", which adds/swaps/deletes edges, and secures that the resulting graph is again cycle-free (so as to remain in the realm of DAGs).

Start: Take a DAG G_0 and calculate its probability $P(G_0|D)$.
$t \leftarrow 0$

While (there is still coffee in the mug):
$G_{t'} \leftarrow modify(G_t)$
Accept new DAG with probability

$$\min(1, \frac{P(G_{t'}|D)}{P(G_t|D)}).$$

If accepted $G_{t+1} \leftarrow G_{t'}$ and $t \leftarrow t+1$.
end while.
Output G_t.

Theorem: This algorithm converges (after a lot of coffee) to the optimal G.

Why?: With the suggested acceptance probability, the accept/reject decision defines a Markov chain on the space of DAGs. By applying steps of this Markov chain one ensure convergence to the MC's equilibrium distribution. This means that after a burn-in phase, the algorithm explores this distribution, and thus samples the more likely DAGs more often.

https://en.wikipedia.org/wiki/Bayesian_inference_in_phylogeny#/media/File:Robot_metaphor.png

Remark: In reality, MCMC is better for integration than for optimization.

It is better in the end not to select one DAG, but to look for frequently used edges.

Problem: A graph that is made up of edges occurring frequently in the MCMC run need not be acyclic.

Back to the problem of small sample sizes. It is hard to believe that with little data we can estimate so much detail.

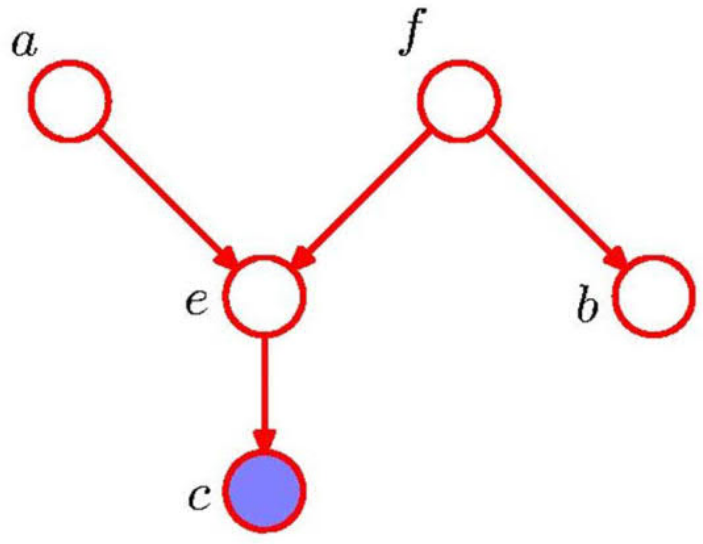
The MCMC algorithm explores across the equilibrium distribution of the MC in DAG space. We can therefore ask about the stability of certain, interesting features.

Example: What is the probability of having gene x directly linked to gene y. Answer: Run MCMC for a long time. Count the frequency at which this holds true.

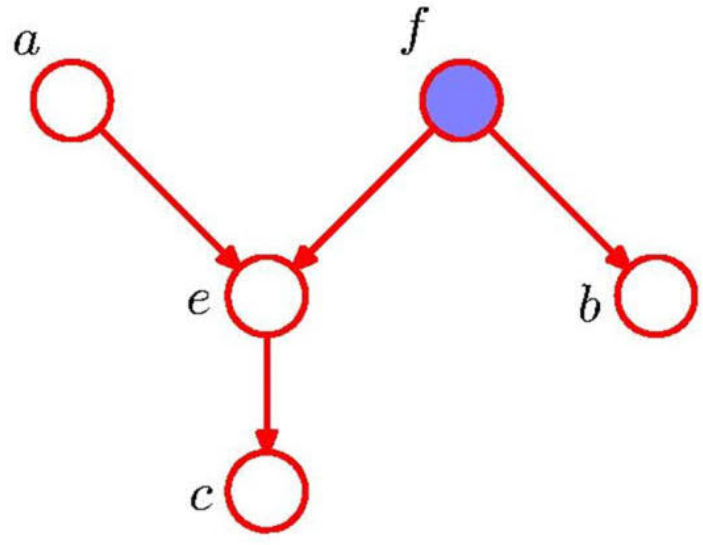
D-separation

- A, B, and C are non-intersecting subsets of nodes in a directed graph.
- A path from A to B is blocked if it contains a node such that either
  a) the arrows on the path meet either head-to-tail or tail-to-tail at the node, and the node is in the set C, or
  b) the arrows meet head-to-head at the node, and neither the node, nor any of its descendants, are in the set C.
- If all paths from A to B are blocked, A is said to be d-separated from B by C.
- If A is d-separated from B by C, the joint distribution over all variables in the graph satisfies $A \perp\!\!\!\perp B \mid C$.

D-separation: Example



$a \not\perp\!\!\!\perp b \mid c$



$a \perp\!\!\!\perp b \mid f$

d-separation

Alternative phrasing (from "Causation, prediction, and search"):

Following Pearl (1988), we say that for a graph G, if X and Y are vertices in G, $X \neq Y$, and W is a set of vertices in G not containing X or Y, then X and Y are d-separated given W in G if and only if there exists no undirected path U between X and Y, such that (i) every collider on U has a descendent in W and (ii) no other vertex on U is in W.

Connection to conditional independence

$A, B \ldots$ nodes
$C \ldots$ set of nodes

If P is a discrete distribution faithful to a graph G, then A and B are d-separated given a set of variables C if and only if A and B are conditionally independent given C.

If P is a distribution linearly faithful to a graph G, then A and B are d-separated given C if and only if $\rho_{A,B|C} = 0$.
(Spirtes, Causation, Prediction, and Search)

Heuristics to compute BNs: PC algorithm

See "Causation, Prediction, and Search" by Spirtes, Glymour, and Scheines, p116ff. Roughly it works as follows:

- Iterative algorithm
- Starts with complete, undirected graph
- Find edges which are conditionally independent with respect to 1 other variable. Delete.
- Find edges which are CI wrt to 2-element-sets of variables. Delete.
- etc.
- Find and orient the colliders

Issues

- Why do we delete these edges? Because if a BN would contain such a CI edge, this would contradict the directed local Markov condition (A variable X_i is conditionally independent of its "non-descendants" given the parents of X_i.)
- How to test for conditional independence? Partial correlation!
- Does it depend on the order of processing? No and yes.
- How does one orient the edges?

by any subset of variables containing *Y* but not *X*, *Z*, the algorithm will mistakenly require a collision at *Y*, and this requirement will ramify through orientations of other edges. Or, if the true structure contains a collision at *Y* but *X* - *Y* is omitted in the input to step C), no unique orientation will be given to *Y* - *Z*, and this uncertainty may ramify through the orientations of other edges on paths including *Z*.

Instabilities may also arise in Step C) because of errors in the list of d-separation relations input, even when the underlying undirected graph is correct. If in the input to C), *X* is adjacent to *Y* and *Y* to *Z* but not *X* to *Z* and a d-separation relation between *X* and *Z* given **S** containing *Y* is omitted from the input, no orientation error will result unless no other set containing *Y* d-separates *X* and *Z*. But if in the true directed graph, the edges between *X* and *Y* and between *Y* and *Z* collide at *Y*, and a d-separation relation involving *X* and *Z* and some set **U** containing *Y* but not *X* or *Z* is erroneously included in the input, the algorithm will conclude that there is no collision at *Y*, and this error may be ramified to other edges.

A little reflection on Step C) reveals that its output may not be a collection of directed acyclic graphs if one of the four assumptions listed at the beginning of this section is violated. This is not necessarily a defect of the algorithm. If the algorithm finds that the edges *X* - *Y* - *Z* collide at *Y*, and *Y* - *Z* - W collide at *Z*, it will create a pattern with an edge *Y* <-> *Z*. Double headed edges can occur when the causal structure is not causally sufficient, or when there is an error in input (as from sampling variation). They have a theoretical role in identifying the presence of unmeasured common causes, an issue discussed further in the next chapter.

5.4.2 The PC Algorithm

In the worst case, the SGS algorithm requires a number of d-separation tests that increases exponentially with the number of vertices, as must any algorithm based on conditional independence relations or vanishing partial correlations. But the SGS algorithm is very inefficient because for edges in the true graph the worst case is also the expected case. For any undirected edge that is in the graph *G*, the number of d-separation tests that must be conducted in stage B) of the algorithm is unaffected by the connectivity of the true graph, and therefore even for sparse graphs the algorithm rapidly becomes infeasible as the number of vertices increases. Besides problems of computational feasibility, the algorithm has problems of reliability when applied to sample data. The determination of higher order conditional independence relations from sample distributions is generally less reliable than is the determination of lower order independence relations. With, say, 37 variables taking three values

each, to determine the conditional independence of two variables on the set of all remaining variables requires considering the relations among the frequencies of 3^{35} distinct states, only a fraction of which will be instantiated even in very large samples.

We should like an algorithm that has the same input/output relations as the SGS procedure for faithful distributions but which for sparse graphs does not require the testing of higher order independence relations in the discrete case, and in any case requires testing as few d-separation relations as possible. The following procedure (Spirtes, Glymour, and Scheines, 1991) starts by forming the complete undirected graph, then "thins" that graph by removing edges with zero order conditional independence relations, thins again with first order conditional independence relations, and so on. The set of variables conditioned on need only be a subset of the set of variables adjacent to one or the other of the variables conditioned.

Let **Adjacencies**(C,A) be the set of vertices adjacent to A in directed acyclic graph C. (In the algorithm, the graph C is continually updated, so **Adjacencies**(C,A) is constantly changing as the algorithm progresses.)

PC Algorithm:

A.) Form the complete undirected graph C on the vertex set **V**.
B.)
$n = 0$.
repeat
repeat
select an ordered pair of variables X and Y that are adjacent in C such that **Adjacencies**(C,X)\{Y} has cardinality greater than or equal to n, and a subset **S** of **Adjacencies**(C,X)\{Y} of cardinality n, and if X and Y are d-separated given **S** delete edge X - Y from C and record **S** in **Sepset**(X, Y) and **Sepset**(Y,X);
until all ordered pairs of adjacent variables X and Y such that **Adjacencies**(C, X)\{Y} has cardinality greater than or equal to n and all

C.) For each triple of vertices X, Y, Z such that the pair X, Y and the pair Y, Z are each adjacent in C but the pair X, Z are not adjacent in C, orient X - Y - Z as X -> Y <- Z if and only if Y is not in **Sepset**(X,Z).

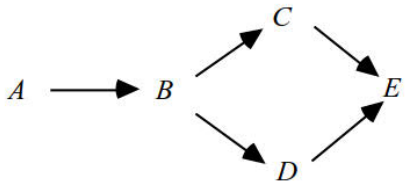
D. repeat

If A -> B, B and C are adjacent, A and C are not adjacent, and there is no arrowhead at B, then orient B - C as B -> C.

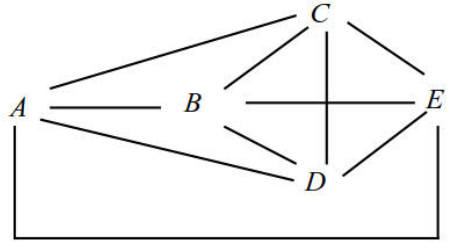
If there is a directed path from A to B, and an edge between A and B, then orient A - B as A -> B.

until no more edges can be oriented.

Figure 1 traces the operation of the first two parts of the PC algorithm:



True Graph



Complete Undirected Graph

---

n = 0 No zero order independencies

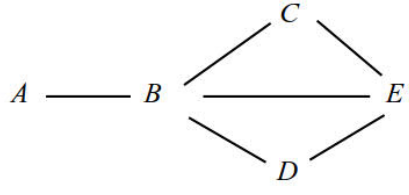
---

n = 1 First order independencies

$A \perp\!\!\!\perp C \mid B$ $A \perp\!\!\!\perp D \mid B$

$A \perp\!\!\!\perp E \mid B$ $C \perp\!\!\!\perp D \mid B$

Resulting Adjacencies



---

n = 2: Second order independencies

Resulting Adjacencies

Although it does not in this case, stage B) of the algorithm may continue testing for some steps after the set of adjacencies in the true directed graph has been identified. The undirected graph at the bottom of figure 1 is now partially oriented in step C). The triples of variables with only two adjacencies among them are:

$A - B - C$; $A - B - D$;
$C - B - D$; $B - C - E$;
$B - D - E$; $C - E - D$

E is not in **Sepset**(C,D) so $C - E$ and $E - D$ collide at E. None of the other triples form colliders. The final pattern produced by the algorithm is shown in figure 2.

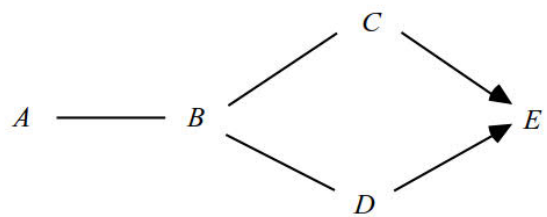


Figure 2

The pattern in figure 2 characterizes a faithful indistinguishability class. Every orientation of the undirected edges in figure 2 is permissible that does not include a collision at B.

5.4.2.1 Complexity

The complexity of the algorithm for a graph G is bounded by the largest degree in G. Let k be the maximal degree of any vertex and let n be the number of vertices. Then in the worst case the number of conditional independence tests required by the algorithm is bounded by

PC algorithm: order of processing

Theorem: If for the given data there exists a BN representation, then the order of processing is unimportant, i.e., any order will recover the correct BN.
Without proof.

For approximation purposes, the order matters.