5 Structure Learning

The previous chapters assumed that the structures of our probabilistic models were known. This chapter discusses methods for learning the structure of models from data.[1] We begin by explaining how to compute the probability of a graphical structure given the data. Generally, we want to maximize this probability. Because the space of possible graphical structures is usually too large to enumerate, we discuss ways to search this space efficiently.

[1] Overviews of Bayesian network structure learning can be found in the following textbooks: D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009. R. E. Neapolitan, *Learning Bayesian Networks*. Prentice Hall, 2003.

5.1 Bayesian Network Scoring

We want to be able to score a network structure G based on how well it models the data. A maximum a posteriori approach to structure learning involves finding a G that maximizes $P(G \mid D)$. We first explain how to compute a *Bayesian network score* based on $P(G \mid D)$ to measure how well G models the data. We then explain how to go about searching the space of networks for the highest scoring network. Like inference in Bayesian networks, it can be shown that for general graphs and input data, learning the structure of a Bayesian network is NP-hard.[2]

[2] See D. M. Chickering, ''Learning Bayesian Networks is NP-Complete,'' in *Learning from Data: Artificial Intelligence and Statistics V*, D. Fisher and H.-J. Lenz, eds., Springer, 1996, pp. 121–130. D. M. Chickering, D. Heckerman, and C. Meek, ''Large-Sample Learning of Bayesian Networks is NP-Hard,'' *Journal of Machine Learning Research*, vol. 5, pp. 1287–1330, 2004.

We compute $P(G \mid D)$ using Bayes' rule and the law of total probability:

$$P(G \mid D) \propto P(G)P(D \mid G) \tag{5.1}$$

$$= P(G) \int P(D \mid \theta, G) p(\theta \mid G) \, \mathrm{d}\theta \tag{5.2}$$

Integrating with respect to θ results in[3]

$$P(G \mid D) = P(G) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij0})}{\Gamma(\alpha_{ij0} + m_{ij0})} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ijk} + m_{ijk})}{\Gamma(\alpha_{ijk})} \tag{5.3}$$

[3] For the derivation, see the appendix of G. F. Cooper and E. Herskovits, ''A Bayesian Method for the Induction of Probabilistic Networks from Data,'' *Machine Learning*, vol. 4, no. 9, pp. 309–347, 1992.

In the equation above,

$$\alpha_{ij0} = \sum_{k=1}^{r_i} \alpha_{ijk} \qquad m_{ij0} = \sum_{k=1}^{r_i} m_{ijk} \tag{5.4}$$

Finding the G that maximizes equation (5.2) is the same as finding the G that maximizes what is called the *Bayesian score*:

$$\log P(G \mid D) = \log P(G) + \sum_{i=1}^{n} \sum_{j=1}^{q_i} \left(\log \left(\frac{\Gamma(\alpha_{ij0})}{\Gamma(\alpha_{ij0} + m_{ij0})} \right) + \sum_{k=1}^{r_i} \log \left(\frac{\Gamma(\alpha_{ijk} + m_{ijk})}{\Gamma(\alpha_{ijk})} \right) \right) \tag{5.5}$$

The Bayesian score is more convenient to compute numerically because it is easier to add the logarithm of small numbers together than to multiply small numbers together. Many software libraries can compute the logarithm of the gamma function directly.

A variety of different graph priors have been explored in the literature, although a uniform prior is often used in practice, in which case $\log P(G)$ can be dropped from the computation of the Bayesian score in equation (5.5). Algorithm 5.1 provides an implementation.

```
function bayesian_score_component(M, α)
    p =  sum(loggamma.(α + M))
    p -= sum(loggamma.(α))
    p += sum(loggamma.(sum(α,dims=2)))
    p -= sum(loggamma.(sum(α,dims=2) + sum(M,dims=2)))
    return p
end

function bayesian_score(vars, G, D)
    n = length(vars)
    M = statistics(vars, G, D)
    α = prior(vars, G)
    return sum(bayesian_score_component(M[i], α[i]) for i in 1:n)
end
```

Algorithm 5.1. An algorithm for computing the Bayesian score for a list of variables `vars` and a graph `G` given data `D`. This method uses a uniform prior $\alpha_{ijk} = 1$ for all i, j, and k as generated by algorithm 4.2. The `loggamma` function is provided by `SpecialFunctions.jl`. The previous chapter introduced the `statistics` and `prior` functions. Note that $\log(\Gamma(\alpha)/\Gamma(\alpha + m)) = \log\Gamma(\alpha) - \log\Gamma(\alpha + m)$ and that $\log\Gamma(1) = 0$.

A byproduct of optimizing the structure with respect to the Bayesian score is that we are able to find the right balance in the model complexity given the available data. We do not want a model that misses out on capturing important relationships between variables, but we also do not want a model that has too many parameters to be adequately learned from limited data.

To illustrate how the Bayesian score helps us balance model complexity, consider the network in figure 5.1. The value of A weakly influences the value of B, and C is independent of the other variables. We sample from this ''true'' model to generate data D and then try to learn the model structure. There are 25 possible network structures involving three variables, but we will focus on the scores for the models in figure 5.2.

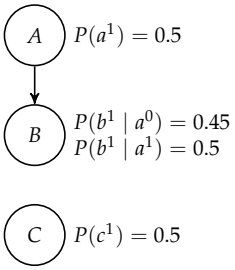


Figure 5.1. A simple Bayesian network to illustrate how the Bayesian network score helps us balance model complexity.

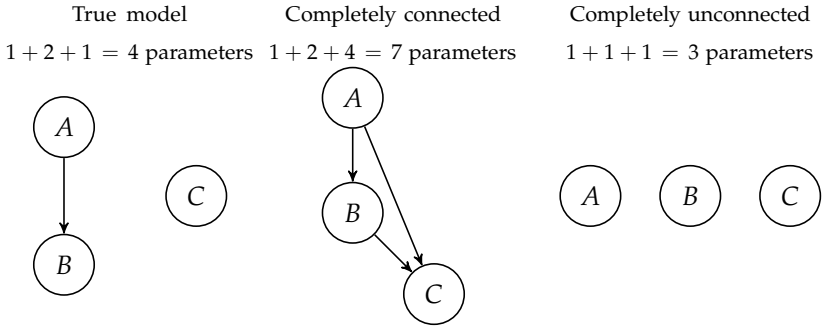


Figure 5.2. Three Bayesian network structures with varying levels of complexity.

Figure 5.3 shows how the Bayesian scores of the completely connected and unconnected models compare to the true model as the amount of data increases. In the plot, we subtract the score of the true model, so values above 0 indicate that the model provides a better representation than the true model given the available data. The plot shows that the unconnected model does better than the true model when there are fewer than 5×10^3 samples. The completely connected model never does better than the true model, but it starts to do better than the unconnected model at about 10^4 samples because there are sufficient data to adequately estimate its seven independent parameters.

5.2 Directed Graph Search

In *directed graph search*, we search the space of directed acyclic graphs for one that maximizes the Bayesian score. The space of possible Bayesian network structures grows superexponentially.[4] With 10 nodes, there are 4.2×10^{18} possible directed acyclic graphs. With 20 nodes, there are 2.4×10^{72}. Except for Bayesian networks with few nodes, we cannot enumerate the space of possible structures to find the highest scoring network. Therefore, we have to rely on a search strategy.

[4] R. W. Robinson, ''Counting Labeled Acyclic Digraphs,'' in *Ann Arbor Conference on Graph Theory*, 1973.

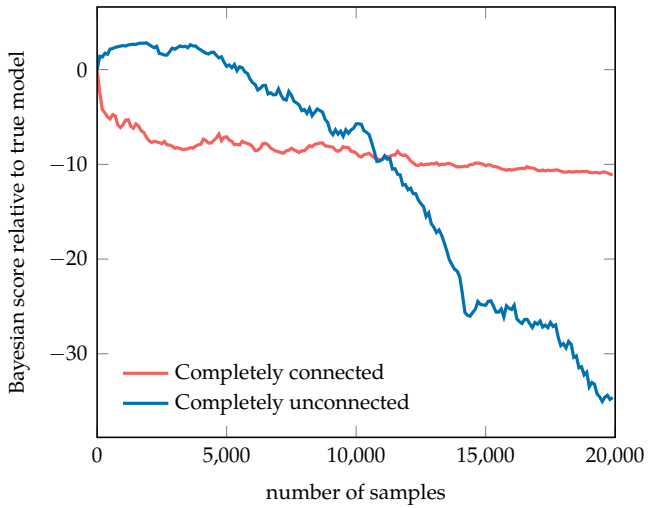


Figure 5.3. Bayesian network structure learning balances model complexity with the available data. The completely connected model never outperforms the true model, whereas the completely unconnected model eventually underperforms once more than about 5×10^3 samples have been drawn. This result indicates that simpler models can outperform complicated models when data is scarce—even when a more complicated model generated the samples.

Fortunately, search is a general problem, and a wide variety of different generic search algorithms have been studied over the years.

One of the most common search strategies is called *K2*.[5] The search (algorithm 5.2) runs in polynomial time but does not guarantee finding a globally optimal network structure. It can use any scoring function, but it is often used with the Bayesian score because of its ability to balance the complexity of the model with the amount of data available. K2 begins with a graph with no directed edges and then iterates over the variables according to a provided ordering, greedily adding parents to the nodes in a way that maximally increases the score. It is common for K2 to impose an upper bound on the number of parents for any one node to reduce the required computation. The original K2 algorithm assumed a unit uniform Dirichlet prior with $\alpha_{ijk} = 1$ for all i, j, and k, but any prior can be used in principle.

A general search strategy is *local search*, which is sometimes called *hill climbing*. Algorithm 5.3 provides an implementation. We start with an initial graph and then move to the highest scoring neighbor. The neighborhood of a graph consists of the graphs that are only one basic graph operation away, where the basic graph operations include introducing an edge, removing an edge, and reversing an edge. Of course, not all operations are possible from a particular graph, and operations

[5] The name comes from the fact that it is an evolution of a system called Kutató. The algorithm was introduced by G. F. Cooper and E. Herskovits, "A Bayesian Method for the Induction of Probabilistic Networks from Data," *Machine Learning*, vol. 4, no. 9, pp. 309–347, 1992.

```
struct K2Search
    ordering::Vector{Int} # variable ordering
end

function fit(method::K2Search, vars, D)
    G = SimpleDiGraph(length(vars))
    for (k,i) in enumerate(method.ordering[2:end])
        y = bayesian_score(vars, G, D)
        while true
            y_best, j_best = -Inf, 0
            for j in method.ordering[1:k]
                if !has_edge(G, j, i)
                    add_edge!(G, j, i)
                    y′ = bayesian_score(vars, G, D)
                    if y′ > y_best
                        y_best, j_best = y′, j
                    end
                    rem_edge!(G, j, i)
                end
            end
            if y_best > y
                y = y_best
                add_edge!(G, j_best, i)
            else
                break
            end
        end
    end
    return G
end
```

Algorithm 5.2. K2 search of the space of directed acyclic graphs using a specified variable ordering. This variable ordering imposes a topological ordering in the resulting graph. The `fit` function takes an ordered list variables `vars` and a dataset `D`. The method starts with an empty graph and iteratively adds the next parent that maximally improves the Bayesian score.

that introduce cycles into the graph are invalid. The search continues until the current graph scores no lower than any of its neighbors.

An *opportunistic* version of local search is implemented in algorithm 5.3. Rather than generating all graph neighbors at every iteration, this method generates a single random neighbor and accepts it if its Bayesian score is greater than that of the current graph.

```
struct LocalDirectedGraphSearch
    G     # initial graph
    k_max # number of iterations
end

function rand_graph_neighbor(G)
    n = nv(G)
    i = rand(1:n)
    j = mod1(i + rand(2:n)-1, n)
    G′ = copy(G)
    has_edge(G, i, j) ? rem_edge!(G′, i, j) : add_edge!(G′, i, j)
    return G′
end

function fit(method::LocalDirectedGraphSearch, vars, D)
    G = method.G
    y = bayesian_score(vars, G, D)
    for k in 1:method.k_max
        G′ = rand_graph_neighbor(G)
        y′ = is_cyclic(G′) ? -Inf : bayesian_score(vars, G′, D)
        if y′ > y
            y, G = y′, G′
        end
    end
    return G
end
```

Algorithm 5.3. Local directed graph search, which starts with an initial directed graph `G` and opportunistically moves to a random graph neighbor whenever its Bayesian score is greater. It repeats this process for `k_max` iterations. Random graph neighbors are generated by either adding or removing a single edge. Edge addition can result in a graph with cycles, in which case we assign a score of $-\infty$.

Local search can get stuck in *local optima*, preventing it from finding the globally optimal network structure. Various strategies have been proposed for addressing local optima, including the following:[6]

- *Randomized restart*. Once a local optima has been found, simply restart the search at a random point in the search space.
- *Simulated annealing*. Instead of always moving to the neighbor with greatest fitness, the search can visit neighbors with lower fitness according to some randomized exploration strategy. As the search progresses, the randomness in

[6] The field of optimization is quite vast, and many methods have been developed for addressing local optima. This textbook provides an overview: M. J. Kochenderfer and T. A. Wheeler, *Algorithms for Optimization*. MIT Press, 2019.

the exploration decreases according to some schedule. This approach is called simulated annealing because of its inspiration from annealing in metallurgy.

- *Genetic algorithms*. The procedure begins with an initial random population of points in the search space represented as binary strings. Each bit in a string indicates the presence or absence of an arrow between two nodes. String manipulation thus allows for searching the space of directed graphs. The individuals in the population reproduce at a rate proportional to their score. Individuals selected for reproduction have their strings recombined randomly through genetic crossover. Genetic crossover involves selecting a crossover point on two randomly selected individuals and then swapping the strings after that point. Mutations are also introduced randomly into the population by randomly flipping bits in the strings. The process of evolution continues until a satisfactory point in the search space is found.

- *Memetic algorithms*. This approach is sometimes called *genetic local search* and is simply a combination of genetic algorithms and local search. After genetic recombination, local search is applied to the individuals.

- *Tabu search*. Previous methods can be augmented to maintain a *tabu list* containing recently visited points in the search space. The search algorithm avoids neighbors in the tabu list.

Some search strategies may work better than others on certain datasets, but in general finding the global optima remains NP-hard. Many applications, however, do not require the globally optimal network structure. A locally optimal structure is often acceptable.

5.3 Markov Equivalence Classes

As discussed earlier, the structure of a Bayesian network encodes a set of conditional independence assumptions. An important observation to make when trying to learn the structure of a Bayesian network is that two different graphs can encode the same independence assumptions. As a simple example, the two-variable network $A \rightarrow B$ has the same independence assumptions as $A \leftarrow B$. Solely on the basis of the data, we cannot justify the direction of the edge between A and B.

If two networks encode the same conditional independence assumptions, we say they are *Markov equivalent*. It can be proven that two graphs are Markov equivalent if and only if they have (1) the same edges without regard to direction and (2) the same *immoral v-structure*. An immoral v-structure is a v-structure $X \to Y \leftarrow Z$ with X and Z not directly connected, as shown in figure 5.4. A *Markov equivalence class* is a set containing all the directed acyclic graphs that are Markov equivalent to each other. A method for checking Markov equivalence is given in algorithm 5.4.

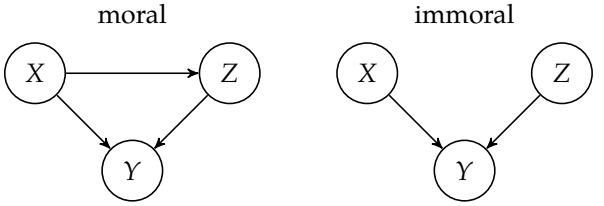


Figure 5.4. Moral and immoral v-structures.

In general, two structures belonging to the same Markov equivalence class may be given different scores. However, if the Bayesian score is used with Dirichlet priors such that $\kappa = \sum_j \sum_k \alpha_{ijk}$ is constant for all i, then two Markov equivalent structures are assigned the same score.[7] Such priors are called *BDe*, and a special case is the *BDeu* prior,[8] which assigns $\alpha_{ijk} = \kappa/(q_i r_i)$. Although the commonly used uniform prior $\alpha_{ijk} = 1$ does not always result in identical scores being assigned to structures in the same equivalence class, they are often fairly close. A scoring function that assigns the same score to all structures in the same class is called *score equivalent*.

[7] This was shown by D. Heckerman, D. Geiger, and D. M. Chickering, "Learning Bayesian Networks: The Combination of Knowledge and Statistical Data," *Machine Learning*, vol. 20, no. 3, pp. 197–243, 1995.

[8] W. L. Buntine, "Theory Refinement on Bayesian Networks," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 1991.

5.4 Partially Directed Graph Search

A Markov equivalence class can be represented as a *partially directed graph*, sometimes called an *essential graph* or a *directed acyclic graph pattern*. A partially directed graph can contain both directed edges and undirected edges. An example of a partially directed graph that encodes a Markov equivalence class is shown in figure 5.5. A directed acyclic graph G is a member of the Markov equivalence class encoded by a partially directed graph G' if and only if G has the same edges as G' without regard to direction and has the same immoral v-structures as G'.

```
function are_markov_equivalent(G, H)
    if nv(G) != nv(H) || ne(G) != ne(H) ||
        !all(has_edge(H, e) || has_edge(H, reverse(e))
                                        for e in edges(G))
        return false
    end
    for c in 1:nv(G)
        parents = inneighbors(G, c)
        for (a, b) in subsets(parents, 2)
            if !has_edge(G, a, b) && !has_edge(G, b, a) &&
               !(has_edge(H, a, c) && has_edge(H, b, c))
                return false
            end
        end
    end
    return true
end
```

Algorithm 5.4. A method for determining whether the directed acyclic graphs `G` and `H` are Markov equivalent. The `subsets` function from `IterTools.jl` returns all subsets of a given set and a specified size.

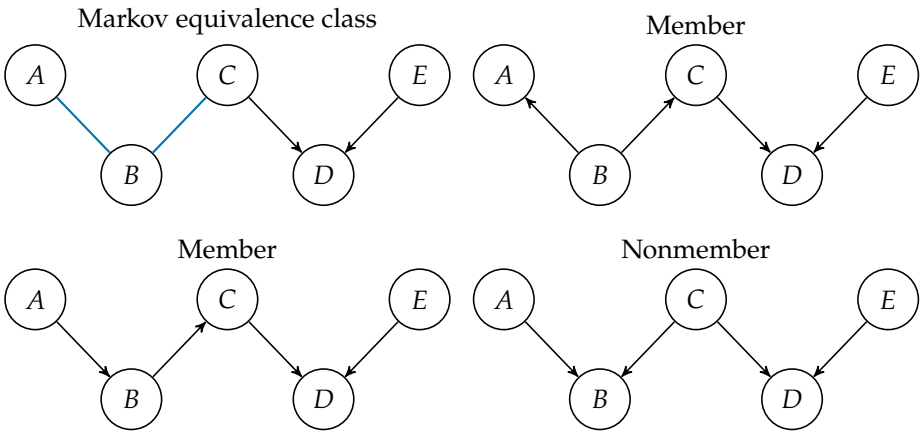


Figure 5.5. Markov equivalence class and examples of members and a nonmember. The nonmember does not belong to the Markov equivalence class because it introduces an immoral v-structure $A \rightarrow B \leftarrow C$ that is not indicated in the partially directed graph.

Instead of searching the space of directed acyclic graphs, we can search the space of Markov equivalence classes represented by partially directed graphs.[9] Although the space of Markov equivalence classes is, of course, smaller than the space of directed acyclic graphs, it is not significantly smaller; the ratio of directed acyclic graphs to equivalence classes asymptotes to around 3.7 fairly quickly.[10] A problem with hill climbing in the space of directed acyclic graphs is that the neighborhood may consist of other graphs that are in the same equivalence class with the same score, which can lead to the search becoming stuck in a local optimum. Searching the space of equivalence classes allows us to jump to different directed acyclic graphs outside of the current equivalence class.

[9] Details of how to to search this space is provided by D. M. Chickering, "Learning Equivalence Classes of Bayesian-Network Structures," *Journal of Machine Learning Research*, vol. 2, pp. 445–498, 2002.

[10] S. B. Gillispie and M. D. Perlman, "The Size Distribution for Markov Equivalence Classes of Acyclic Digraph Models," *Artificial Intelligence*, vol. 141, no. 1–2, pp. 137–155, 2002.

Any of the general search strategies presented in section 5.2 can be used. If a form of local search is used, then we need to define the local graph operations that define the neighborhood of the graph. Examples of local graph operations include:

- If an edge between X and Y does not exist, add either $X - Y$ or $X \to Y$.
- If $X - Y$ or $X \to Y$, then remove the edge between X and Y.
- If $X \to Y$, then reverse the direction of the edge to get $X \leftarrow Y$.
- If $X - Y - Z$, then add $X \to Y \leftarrow Z$.

To score a partially directed graph, we generate a member of its Markov equivalence class and compute its score.

5.5 Summary

- Fitting a Bayesian network to data requires selecting the Bayesian network structure that dictates the conditional dependencies between variables.
- Bayesian approaches to structure learning maximize the Bayesian score, which is related to the probability of the graph structure given a dataset.
- The Bayesian score promotes simpler structures for smaller datasets and supports more complicated structures for larger datasets.
- The number of possible structures is superexponential in the number of variables, and finding a structure that maximizes the Bayesian score is NP-hard.

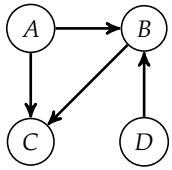
- Directed graph search algorithms like K2 and local search can be efficient but do not guarantee optimality.
- Methods like partially directed graph search traverse the space of Markov equivalence classes, which may be more efficient than searching the larger space of directed acyclic graphs.

5.6 Exercises

Exercise 5.1. How many neighbors does an edgeless directed acyclic graph with m nodes have?

Solution: Of the three basic graph operations, we can only add edges. We can add any edge to an edgeless directed acyclic graph and it will remain acyclic. There are $m(m-1) = m^2 - m$ node pairs, and therefore that many neighbors.

Exercise 5.2. How many networks are in the neighborhood of the following Bayesian network?

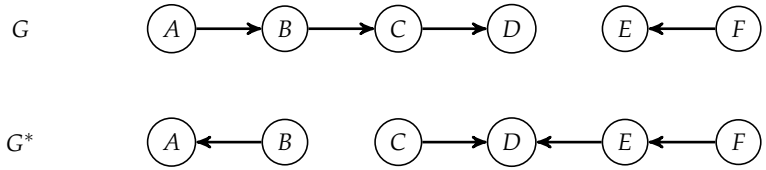


Solution: We can perform the following graph operations:

- add $A \to D$, $D \to A$, $D \to C$
- remove $A \to B$, $A \to C$, $B \to C$, $D \to B$
- flip $A \to B$, $B \to C$, $D \to B$

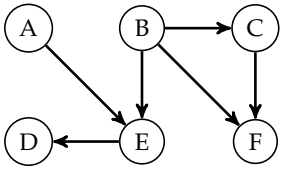
Thus, there are 10 Bayesian networks in the neighborhood.

Exercise 5.3. Suppose we start local search with Bayesian network G. What is the fewest number of iterations of local search that could be performed to converge to the optimal Bayesian network G^*?

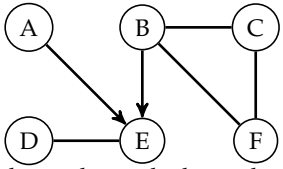


Solution: At each iteration, local search can move from the original network to a network in its neighborhood, which is at most one edge operation from the original network. Since there are three differences between the edges of G and $G*$, performing local search from G would require a minimum of three iterations to arrive at $G*$. One potential minimal sequence of local search iterations could be flipping $A \to B$, removing $B \to C$, and adding $E \to D$. We assume the graphs formed with these edge operations yielded the highest Bayesian scores of all graphs in the considered neighborhood.

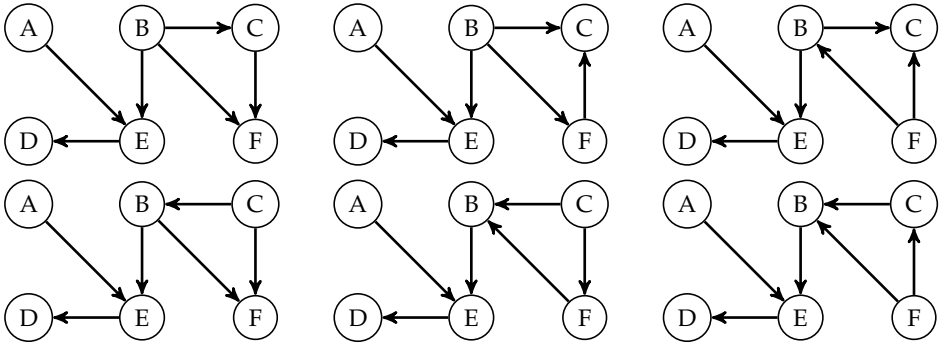
Exercise 5.4. Draw the partially directed acyclic graph representing the Markov equivalence class of the following Bayesian network. How many graphs are in this Markov equivalence class?



Solution: The Markov equivalence class can be represented with the following partially directed acyclic graph:

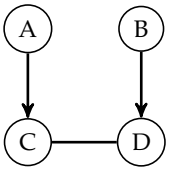


There are six networks in this Markov equivalence class, which are shown below:



Exercise 5.5. Give an example of a partially directed acyclic graph with four nodes that does not define a (non-empty) Markov equivalence class.

Solution: Consider the following partially directed acyclic graph:



We cannot replace the undirected edge with a directed edge because doing so would introduce a new v-structure.