chapter7

# Chapter 7[¶](#Chapter-7)

# Probabilistic Inference with Join Tree Clustering[¶](#Xf2f6c5858e92fa25f457704f5bd095deb2858de)

## Graph Layout and Drawing[¶](#Graph-Layout-and-Drawing)

The following two functions use **Graphviz** to perform the graph layout and drawing tasks for directed acyclic and undirected graphs.

### dagviz(nodes, edges, name)[¶](#dagviz(nodes,-edges,-name))

This function takes the **nodes** and **edges** for a directed acyclic graph, does the graph layout, creates a *dot* and *PNG* files for drawing the DAG, and writes them in files with the given **name**.

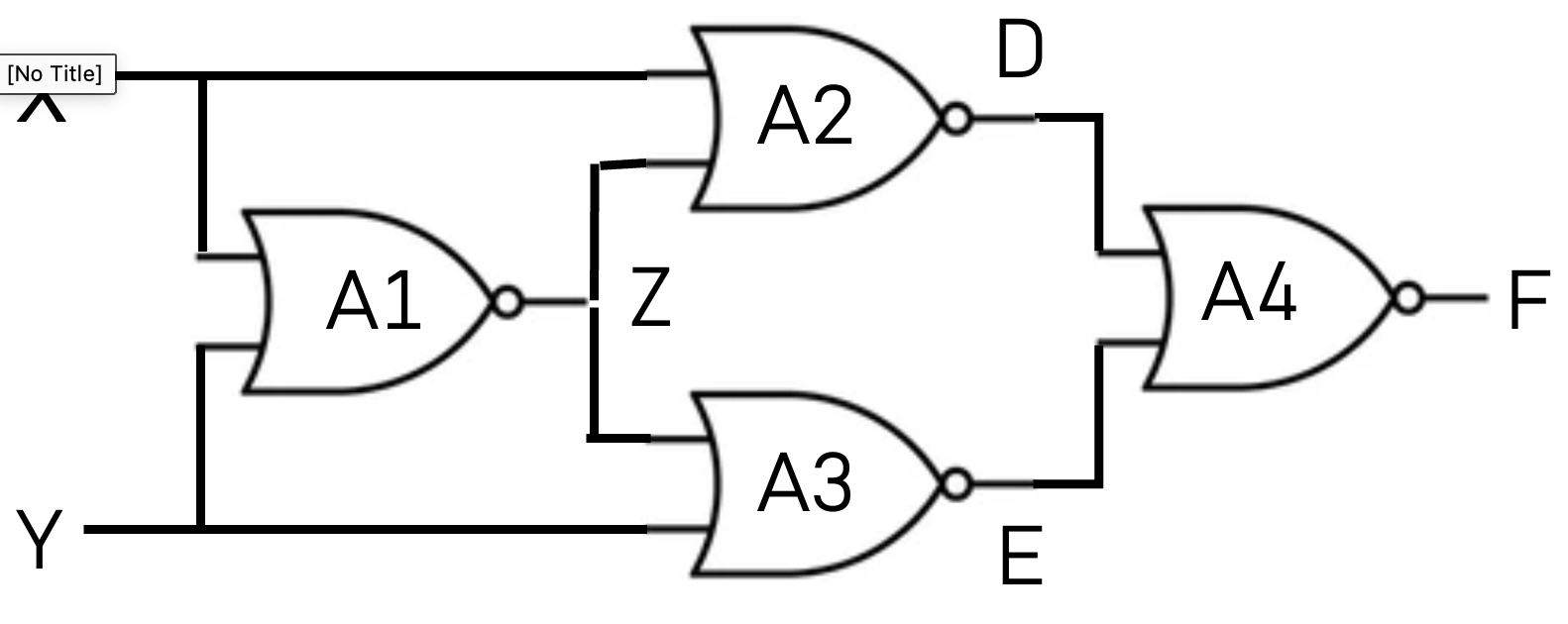
### gviz(nodes, edges, name)[¶](#gviz(nodes,-edges,-name))

This function takes the **nodes** and e**dges** for an undirected graph, does the graph layout, creates a *dot* and *PNG* files for drawing the undirected graph, and writes them in files with the given **name**.

In [1]:

import pygraphviz as pgv  
  
  
def dagviz(nodes, edges, name):  
 G = pgv.AGraph(strict=False, directed=True)  
 G.add\_nodes\_from(nodes)  
 for edge in edges:  
 G.add\_edge(edge)  
 G.write(name + ".dot")  
 # use dot  
 G.layout(prog="dot")  
 # write previously positioned graph to PNG file  
 G.draw(name + ".png")  
  
  
def gviz(nodes, edges, name):  
 A = pgv.AGraph()  
 A.add\_nodes\_from(nodes)  
 for edge in edges:  
 A.add\_edge(edge)  
 A.write(name + ".dot")  
 # use dot  
 A.layout(prog="dot")  
 # write previously positioned graph to PNG file  
 A.draw(name + ".png")

## Circuit diagnosis Bayesian Network[¶](#Circuit-diagnosis-Bayesian-Network)



#### Create the BN model and specify its structure[¶](#X96f3545908a45fd69c3e6a132c8c4cc4b3e0e7b)

The following code creates the circuit diagnosis BN model. The model factors according to a DAG having 10 nodes and 12 edges.

The nodes are the following network variables:

A1, A3, Y. A2, X, Z, E, D, A4, F

Four of the above variables network variables

A1, A2, A3, A4

represent hypotheses about the working condition of respective NOR gates of the circuit.

All network variables are binary, taking values 0 and 1.

The directed edges of the DAG are as follows:

A1 -> Z, A3 -> E, Y -> Z , Y -> E, A2 -> D,   
X -> Z, X -> D, Z -> E, Z -> D, E -> F,   
D -> F, A4 -> F

## Qualitative Specification of the Bayesian Network[¶](#X03a606d18794c9f19f42f42ba4331fe43b10526)

The following code creates the qualitative component of the circuit diagnosis Bayesian network model.

### 1. Specifying the nodes and directed edges of the Bayesian network[¶](#X25a715f5fbe164d711d5ff8754701b9c9571181)

The following code specifies the nodes which are the network variables and the directed edges which represents direct probabilistic influence between distinct pairs of network variables.

In [2]:

from pgmpy.models import BayesianNetwork  
def circuit():  
 circuit: BayesianNetwork = BayesianNetwork()  
 circuit.add\_nodes\_from(["X", "Y", "Z", "D", "E", "F", "A1", "A2", "A3", "A4"])  
 circuit.add\_edge("X", "Z")  
 circuit.add\_edge("Y", "Z")  
 circuit.add\_edge("A1", "Z")  
 circuit.add\_edge("X", "D")  
 circuit.add\_edge("Z", "D")  
 circuit.add\_edge("A2", "D")  
 circuit.add\_edge("Z", "E")  
 circuit.add\_edge("Y", "E")  
 circuit.add\_edge("A3", "E")  
 circuit.add\_edge("D", "F")  
 circuit.add\_edge("A4", "F")  
 circuit.add\_edge("E", "F")  
 return circuit

### 2. Creating a directed acyclic graph representing the Bayesian network model[¶](#X1909bfda9b286eb258ccd5a4f8d84148cac77af)

directed acyclic graph object to represent the BN structure having the nodes and edges specified for the BN model.

In [3]:

from pgmpy.base import DAG  
def circuit\_dag():  
 model = circuit()  
 G = DAG()  
 nodes = model.nodes  
 edges = model.edges  
 G.add\_nodes\_from(nodes)  
 G.add\_edges\_from(edges)  
 return G  
  
print(f"creating the directed acyclic graph for the circuit diagnosis example")  
G = circuit\_dag()  
print(G)

creating the directed acyclic graph for the circuit diagnosis example  
DAG with 10 nodes and 12 edges

## Creating DOT and PNG files for the DAG of the circuit diagnosis BN[¶](#X5f4077c194cff42014a4bca726fa7b90fe08bed)

The following code uses our **dagviz()** function to create the **DOT** and **PNG** files named *DAG\_Circuit* for the circuit diagnosis BN. Note that the nodes and edges parameters for **dagviz()** are inputted from the nodes and edges of the DAG object *G*.

In [4]:

dagviz(G.nodes, G.edges, "DAG\_circuit")

## Getting the minimal d-separator set between two nodes[¶](#X414e346e8061e5b39cade17652b5c397de65906)

The following code uses the **minimal\_dseparator()** method of the **DAG** object, *G*, to get the minimal d-separator set separating the node "E" from the node "X." The two variables for the two nodes are conditionally independent given the values of the variables in a d-separator set.

In [5]:

G.minimal\_dseparator("E", "X")

Out[5]:

{'Y', 'Z'}

## Finding all independencies in the circuit diagnosis Bayesian Network[¶](#X294e4a650259b6f5de2fd5bfccac8cf4a5500a5)

The following code uses the **get\_independencies()** method of the **DAG** object, *G*, to compute all the d-separations in *G*, which implies conditional independencies in the probabilistic model. The result is written in a text file.

In [6]:

from contextlib import redirect\_stdout  
  
with open("circuit\_dag\_d-separations.txt", "w") as f:  
 with redirect\_stdout(f):  
 print(G.get\_independencies())  
f.close()  
print(  
 f"Wrote all d-separations for the circuit DAG into file [circuit\_dag\_d-separations.txt]"  
)

Wrote all d-separations for the circuit DAG into file [circuit\_dag\_d-separations.txt]

## Quantitative Specification of the Bayesian Network[¶](#Xcfb96e0e1c26055db6afb9e78e737028e6b405c)

The following code specifies the quantitative component of the circuit diagnosis Bayesian network model.

The following code specifies the condition probability distributions (**CPD**s) for all the network variables and adds the **CPD**s to the model.

After quantifying the BN model for the circuit diagnosis example, we check the model to see if its specification is correct. The function **check\_model()** should return *True* if the checking is successful.

In [7]:

from pgmpy.factors.discrete import TabularCPD  
model = circuit()  
cpd\_X = TabularCPD(variable="X", variable\_card=2, values=[[0.5], [0.5]])  
cpd\_Y = TabularCPD(variable="Y", variable\_card=2, values=[[0.5], [0.5]])  
cpd\_A1 = TabularCPD(variable="A1", variable\_card=2, values=[[0.95], [0.05]])  
cpd\_A2 = TabularCPD(variable="A2", variable\_card=2, values=[[0.95], [0.05]])  
cpd\_A3 = TabularCPD(variable="A3", variable\_card=2, values=[[0.95], [0.05]])  
cpd\_A4 = TabularCPD(variable="A4", variable\_card=2, values=[[0.95], [0.05]])  
cpd\_Z = TabularCPD(  
 variable="Z",  
 variable\_card=2,  
 values=[  
 [1e-9, 1 - 1e-9, 1 - 1e-9, 1 - 1e-9, 0.5, 0.5, 0.5, 0.5],  
 [1 - 1e-9, 1e-9, 1e-9, 1e-9, 0.5, 0.5, 0.5, 0.5],  
 ],  
 evidence=["A1", "X", "Y"],  
 evidence\_card=[2, 2, 2],  
)  
cpd\_D = TabularCPD(  
 variable="D",  
 variable\_card=2,  
 values=[  
 [1e-9, 1 - 1e-9, 1 - 1e-9, 1 - 1e-9, 0.5, 0.5, 0.5, 0.5],  
 [1 - 1e-9, 1e-9, 1e-9, 1e-9, 0.5, 0.5, 0.5, 0.5],  
 ],  
 evidence=["A2", "X", "Z"],  
 evidence\_card=[2, 2, 2],  
)  
cpd\_E = TabularCPD(  
 variable="E",  
 variable\_card=2,  
 values=[  
 [1e-9, 1 - 1e-9, 1 - 1e-9, 1 - 1e-9, 0.5, 0.5, 0.5, 0.5],  
 [1 - 1e-9, 1e-9, 1e-9, 1e-9, 0.5, 0.5, 0.5, 0.5],  
 ],  
 evidence=["A3", "Y", "Z"],  
 evidence\_card=[2, 2, 2],  
)  
cpd\_F = TabularCPD(  
 variable="F",  
 variable\_card=2,  
 values=[  
 [1e-9, 1 - 1e-9, 1 - 1e-9, 1 - 1e-9, 0.5, 0.5, 0.5, 0.5],  
 [1 - 1e-9, 1e-9, 1e-9, 1e-9, 0.5, 0.5, 0.5, 0.5],  
 ],  
 evidence=["A4", "D", "E"],  
 evidence\_card=[2, 2, 2],  
)  
  
model.add\_cpds(cpd\_X, cpd\_Y, cpd\_Z, cpd\_D, cpd\_E, cpd\_F, cpd\_A1, cpd\_A2, cpd\_A3, cpd\_A4)  
  
print(model.check\_model())

True

## Markov Network Model[¶](#Markov-Network-Model)

The following code uses the method **to\_markov\_model()** for the **BayesianNetwork** object *model* to map the BN model of our circuit diagnosis example to a representation as a Markov network model. The Markov network model will map the **CPD**s of the Bayesian network to **factors**. We use the method **get\_factors()** for the **MarkovNetwork** object, *markovmodel*, and then we print all the factors of the model using a **for** loop.

In [8]:

from pgmpy.models import MarkovNetwork  
markovmodel = model.to\_markov\_model()  
factors = markovmodel.get\_factors()  
for factor in factors:  
 print(factor)

+------+----------+  
| X | phi(X) |  
+======+==========+  
| X(0) | 0.5000 |  
+------+----------+  
| X(1) | 0.5000 |  
+------+----------+  
+------+----------+  
| Y | phi(Y) |  
+======+==========+  
| Y(0) | 0.5000 |  
+------+----------+  
| Y(1) | 0.5000 |  
+------+----------+  
+------+-------+------+------+-----------------+  
| Z | A1 | X | Y | phi(Z,A1,X,Y) |  
+======+=======+======+======+=================+  
| Z(0) | A1(0) | X(0) | Y(0) | 0.0000 |  
+------+-------+------+------+-----------------+  
| Z(0) | A1(0) | X(0) | Y(1) | 1.0000 |  
+------+-------+------+------+-----------------+  
| Z(0) | A1(0) | X(1) | Y(0) | 1.0000 |  
+------+-------+------+------+-----------------+  
| Z(0) | A1(0) | X(1) | Y(1) | 1.0000 |  
+------+-------+------+------+-----------------+  
| Z(0) | A1(1) | X(0) | Y(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| Z(0) | A1(1) | X(0) | Y(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
| Z(0) | A1(1) | X(1) | Y(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| Z(0) | A1(1) | X(1) | Y(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
| Z(1) | A1(0) | X(0) | Y(0) | 1.0000 |  
+------+-------+------+------+-----------------+  
| Z(1) | A1(0) | X(0) | Y(1) | 0.0000 |  
+------+-------+------+------+-----------------+  
| Z(1) | A1(0) | X(1) | Y(0) | 0.0000 |  
+------+-------+------+------+-----------------+  
| Z(1) | A1(0) | X(1) | Y(1) | 0.0000 |  
+------+-------+------+------+-----------------+  
| Z(1) | A1(1) | X(0) | Y(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| Z(1) | A1(1) | X(0) | Y(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
| Z(1) | A1(1) | X(1) | Y(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| Z(1) | A1(1) | X(1) | Y(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
+------+-------+------+------+-----------------+  
| D | A2 | X | Z | phi(D,A2,X,Z) |  
+======+=======+======+======+=================+  
| D(0) | A2(0) | X(0) | Z(0) | 0.0000 |  
+------+-------+------+------+-----------------+  
| D(0) | A2(0) | X(0) | Z(1) | 1.0000 |  
+------+-------+------+------+-----------------+  
| D(0) | A2(0) | X(1) | Z(0) | 1.0000 |  
+------+-------+------+------+-----------------+  
| D(0) | A2(0) | X(1) | Z(1) | 1.0000 |  
+------+-------+------+------+-----------------+  
| D(0) | A2(1) | X(0) | Z(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| D(0) | A2(1) | X(0) | Z(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
| D(0) | A2(1) | X(1) | Z(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| D(0) | A2(1) | X(1) | Z(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
| D(1) | A2(0) | X(0) | Z(0) | 1.0000 |  
+------+-------+------+------+-----------------+  
| D(1) | A2(0) | X(0) | Z(1) | 0.0000 |  
+------+-------+------+------+-----------------+  
| D(1) | A2(0) | X(1) | Z(0) | 0.0000 |  
+------+-------+------+------+-----------------+  
| D(1) | A2(0) | X(1) | Z(1) | 0.0000 |  
+------+-------+------+------+-----------------+  
| D(1) | A2(1) | X(0) | Z(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| D(1) | A2(1) | X(0) | Z(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
| D(1) | A2(1) | X(1) | Z(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| D(1) | A2(1) | X(1) | Z(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
+------+-------+------+------+-----------------+  
| E | A3 | Y | Z | phi(E,A3,Y,Z) |  
+======+=======+======+======+=================+  
| E(0) | A3(0) | Y(0) | Z(0) | 0.0000 |  
+------+-------+------+------+-----------------+  
| E(0) | A3(0) | Y(0) | Z(1) | 1.0000 |  
+------+-------+------+------+-----------------+  
| E(0) | A3(0) | Y(1) | Z(0) | 1.0000 |  
+------+-------+------+------+-----------------+  
| E(0) | A3(0) | Y(1) | Z(1) | 1.0000 |  
+------+-------+------+------+-----------------+  
| E(0) | A3(1) | Y(0) | Z(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| E(0) | A3(1) | Y(0) | Z(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
| E(0) | A3(1) | Y(1) | Z(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| E(0) | A3(1) | Y(1) | Z(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
| E(1) | A3(0) | Y(0) | Z(0) | 1.0000 |  
+------+-------+------+------+-----------------+  
| E(1) | A3(0) | Y(0) | Z(1) | 0.0000 |  
+------+-------+------+------+-----------------+  
| E(1) | A3(0) | Y(1) | Z(0) | 0.0000 |  
+------+-------+------+------+-----------------+  
| E(1) | A3(0) | Y(1) | Z(1) | 0.0000 |  
+------+-------+------+------+-----------------+  
| E(1) | A3(1) | Y(0) | Z(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| E(1) | A3(1) | Y(0) | Z(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
| E(1) | A3(1) | Y(1) | Z(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| E(1) | A3(1) | Y(1) | Z(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
+------+-------+------+------+-----------------+  
| F | A4 | D | E | phi(F,A4,D,E) |  
+======+=======+======+======+=================+  
| F(0) | A4(0) | D(0) | E(0) | 0.0000 |  
+------+-------+------+------+-----------------+  
| F(0) | A4(0) | D(0) | E(1) | 1.0000 |  
+------+-------+------+------+-----------------+  
| F(0) | A4(0) | D(1) | E(0) | 1.0000 |  
+------+-------+------+------+-----------------+  
| F(0) | A4(0) | D(1) | E(1) | 1.0000 |  
+------+-------+------+------+-----------------+  
| F(0) | A4(1) | D(0) | E(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| F(0) | A4(1) | D(0) | E(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
| F(0) | A4(1) | D(1) | E(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| F(0) | A4(1) | D(1) | E(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
| F(1) | A4(0) | D(0) | E(0) | 1.0000 |  
+------+-------+------+------+-----------------+  
| F(1) | A4(0) | D(0) | E(1) | 0.0000 |  
+------+-------+------+------+-----------------+  
| F(1) | A4(0) | D(1) | E(0) | 0.0000 |  
+------+-------+------+------+-----------------+  
| F(1) | A4(0) | D(1) | E(1) | 0.0000 |  
+------+-------+------+------+-----------------+  
| F(1) | A4(1) | D(0) | E(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| F(1) | A4(1) | D(0) | E(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
| F(1) | A4(1) | D(1) | E(0) | 0.5000 |  
+------+-------+------+------+-----------------+  
| F(1) | A4(1) | D(1) | E(1) | 0.5000 |  
+------+-------+------+------+-----------------+  
+-------+-----------+  
| A1 | phi(A1) |  
+=======+===========+  
| A1(0) | 0.9500 |  
+-------+-----------+  
| A1(1) | 0.0500 |  
+-------+-----------+  
+-------+-----------+  
| A2 | phi(A2) |  
+=======+===========+  
| A2(0) | 0.9500 |  
+-------+-----------+  
| A2(1) | 0.0500 |  
+-------+-----------+  
+-------+-----------+  
| A3 | phi(A3) |  
+=======+===========+  
| A3(0) | 0.9500 |  
+-------+-----------+  
| A3(1) | 0.0500 |  
+-------+-----------+  
+-------+-----------+  
| A4 | phi(A4) |  
+=======+===========+  
| A4(0) | 0.9500 |  
+-------+-----------+  
| A4(1) | 0.0500 |  
+-------+-----------+

## Checking the Markov Model is correct[¶](#Checking-the-Markov-Model-is-correct)

The following code uses the **check\_model()** method of the **MarkovModel** object, *markovmodel*, to verify that the model specification is correct.

In [9]:

markovmodel.check\_model()

Out[9]:

True

## Getting the Partition Function for the Markov model[¶](#Xb2f120ab021073e22059779d31702e652b11f39)

The following code uses the **get\_partition\_function()** method of the **MarkovNetwork** object, *markovmodel*, to get the partition function of the Markov model factors. The method returns a number that normalizes the product of all factors to produce a probability distribution when divided by the partition function. A probability distribution must sum up to 1 on all the network variables and all values from their domains.

In [10]:

markovmodel.get\_partition\_function()

Out[10]:

1.0000000000000002

## Getting the nodes and edges of the factor graph[¶](#X57a6505da1b5f7d8aaf3e12c6083c6efa92dcbf)

To get the undirected graph, *factor\_graph*, of the circuit diagnosis example' Markov network model, *markovmodel*, we use the method **to\_factor\_graph()**. We then print the nodes and edges of the undirected graph, factor\_graph. The factor graph consists of two subsets of nodes; one contains the network variables, and the other contains the factors, which are the potential functions of the Markov model.

In [11]:

factor\_graph = markovmodel.to\_factor\_graph()  
print(factor\_graph.nodes)  
print(factor\_graph.edges)

['X', 'Z', 'D', 'Y', 'A1', 'A2', 'E', 'A3', 'F', 'A4', 'phi\_X', 'phi\_Y', 'phi\_Z\_A1\_X\_Y', 'phi\_D\_A2\_X\_Z', 'phi\_E\_A3\_Y\_Z', 'phi\_F\_A4\_D\_E', 'phi\_A1', 'phi\_A2', 'phi\_A3', 'phi\_A4']  
[('X', 'phi\_X'), ('X', 'phi\_Z\_A1\_X\_Y'), ('X', 'phi\_D\_A2\_X\_Z'), ('Z', 'phi\_Z\_A1\_X\_Y'), ('Z', 'phi\_D\_A2\_X\_Z'), ('Z', 'phi\_E\_A3\_Y\_Z'), ('D', 'phi\_D\_A2\_X\_Z'), ('D', 'phi\_F\_A4\_D\_E'), ('Y', 'phi\_Y'), ('Y', 'phi\_Z\_A1\_X\_Y'), ('Y', 'phi\_E\_A3\_Y\_Z'), ('A1', 'phi\_Z\_A1\_X\_Y'), ('A1', 'phi\_A1'), ('A2', 'phi\_D\_A2\_X\_Z'), ('A2', 'phi\_A2'), ('E', 'phi\_E\_A3\_Y\_Z'), ('E', 'phi\_F\_A4\_D\_E'), ('A3', 'phi\_E\_A3\_Y\_Z'), ('A3', 'phi\_A3'), ('F', 'phi\_F\_A4\_D\_E'), ('A4', 'phi\_F\_A4\_D\_E'), ('A4', 'phi\_A4')]

## Creating DOT and PNG files for the undirected factor graph of the circuit diagnosis BN[¶](#Xa61f8bc9b4c980c53fb8d7849131138c5c294eb)

The following code uses our **gviz()** function to create the **DOT** and **PNG** files named *factor\_graph* for the circuit diagnosis Markov model. Note that the nodes and edges parameters for **gviz()** are inputted from the nodes and edges of the undirected factor graph object *factor\_graph*.

In [12]:

gviz(factor\_graph.nodes, factor\_graph.edges, "factor\_graph")

## Using the Pgmpy library to get a join tree from the Markov model[¶](#Xb7d05f0ee32ac663c4183deb1ea181bc84766a3)

The following code gets the junction tree structure built by the **Pgmpy** library by using the **to\_junction\_tree()** method of the Markov model object, *markovmodel*. The code prints the factors for the junction tree model for our circuit diagnosis example.

In [13]:

from pgmpy.models import JunctionTree  
JT = markovmodel.to\_junction\_tree()  
print(JT.factors)

[<DiscreteFactor representing phi(Y:2, X:2, Z:2, E:2, A3:2, A1:2) at 0x168ed71d0>, <DiscreteFactor representing phi(F:2, Z:2, D:2, E:2, A2:2, X:2) at 0x168ed5100>, <DiscreteFactor representing phi(D:2, F:2, A4:2, E:2) at 0x168ed6bd0>]

In [14]:

print(JT.nodes)

[('E', 'Z', 'X', 'Y', 'A3', 'A1'), ('E', 'Z', 'X', 'A2', 'F', 'D'), ('E', 'A4', 'F', 'D')]

## Creating DOT and PNG files for the Pgmpy Join Tree[¶](#X85b4cde4e2e61e26756d79e97ed54a660a31735)

The following code uses our **gviz()** function to create the **DOT** and **PNG** files named *pgmpy\_JT* for the circuit diagnosis Join Tree computed by the Pgmpy library. Note that the nodes and edges parameters for **gviz()** are inputted from the nodes and edges of the **JunctionTree** object *JT*.

In [15]:

gviz(JT.nodes, JT.edges, "pgmpy\_JT")

## Showing the maximum cliques computed by the Pgmpy library[¶](#Xb31545651a987a7d2a6540ea0bf9ac93a7a50a6)

The following code prints the nodes of the junction tree object, *JT*, computed by the **Pgmpy** library. The result shows four maximum cliques compared to the six maximum cliques from our join tree computation. Our join tree clustering computations are superior as they significantly reduce the clique size, which translates to an exponential decrease in the space requirements for making inferences.

**Building the join tree, computed from the join tree clustering algorithm, for the circuit diagnosis BN**.

This process involves creating a join tree structure specific to our Bayesian network model.

We use the **JunctionTree()** constructor to create a join tree structure as an object of the *JunctionTree* class from the *Pgmpy* library.

Then, we add the nodes and edges for the join tree computed from the join tree clustering algorithm for the circuit diagnosis DAG of its Bayesian network model. The nodes are the maximum cliques, and the edges are pairs of nodes connected by an edge in the join tree.

We add the nodes by feeding the list of nodes to the **add\_nodes\_from()** method for the JunctionTree object, *circuit\_JT*. We add the edges by feeding the list of edges to the **add\_edges\_from()** method for the JunctionTree object, *circuit\_JT*.

In [16]:

circuit\_JT = JunctionTree()  
circuit\_JT.add\_nodes\_from([('Z', 'X', 'Y', 'A1'), ('Z', 'X', 'Y', 'E'), ('Z', 'A3', 'Y', 'E'), ('Z', 'X', 'D', 'E'), ('Z', 'X', 'A2', 'D'), ('A4', 'E', 'D', 'F')])  
circuit\_JT.add\_edges\_from([(('Z', 'X', 'Y', 'A1'), ('Z', 'X', 'Y', 'E')),(('Z', 'X', 'Y', 'E'), ('Z', 'A3', 'Y', 'E')),(('Z', 'X', 'Y', 'E'),('Z', 'X', 'D', 'E')),(('Z', 'X', 'D', 'E'),('Z', 'X', 'A2', 'D')),(('Z', 'X', 'D', 'E'),('A4', 'E', 'D', 'F'))])

## Creating DOT and PNG files for the Join Tree from the Tree Clustering Algorithm[¶](#X6ffb107e799c038f854f339a7ea46ea79cf1ad0)

The following code uses our **gviz()** function to create the **DOT** and **PNG** files named *circuit\_JT* for the circuit diagnosis Join Tree computed by the Tree clustering algorithm. Note that the nodes and edges parameters for **gviz()** are inputted from the nodes and edges of the **JunctionTree** object *circuit\_JT*.

In [17]:

gviz(circuit\_JT.nodes, circuit\_JT.edges, "circuit\_JT")

## Getting the Potential Functions of the Markov Model[¶](#X37866ebb9126651ae0a16986beb144ab7b40f45)

The following code maps the CPDs of the Bayesian network, which are *Pgmpy* library type **TabularCPD** objects to *Pgmpy* **DiscreteFactor** objects using the **to\_factor()** method applied to the **TabularCPD** objects.

The code transforms the representation of the circuit BN to a Markov model where the *DiscreteFactor* objects represent the potential functions of the Markov model.

In [18]:

phi\_A1 = cpd\_A1.to\_factor()  
phi\_A2 = cpd\_A2.to\_factor()  
phi\_A3 = cpd\_A3.to\_factor()  
phi\_A4 = cpd\_A4.to\_factor()  
phi\_X = cpd\_X.to\_factor()  
phi\_Y = cpd\_Y.to\_factor()  
phi\_Z = cpd\_Z.to\_factor()  
phi\_D = cpd\_D.to\_factor()  
phi\_E = cpd\_E.to\_factor()  
phi\_F = cpd\_F.to\_factor()

**Computing the Potential Fuctions for the Join Tree**

The following code computes the potential functions for the maximum cliques computed by the tree clustering algorithm for our circuit diagnosis Bayesian network.

The computed join tree has six cliques, which are the following subsets of the Bayesian network variables:

C1: ('Z', 'X', 'Y', 'A1'),  
C2: ('Z', 'X', 'Y', 'E'),  
C3: ('Z', 'A3', 'Y', 'E'),  
C4: ('Z', 'X', 'D', 'E'),  
C5: ('Z', 'X', 'A2', 'D'),  
C6: ('A4', 'E', 'D', 'F')

To compute the potential function for each clique, we multiply all potential functions of the Markov model allocated to the clique. If a clique does not have any potential function allocated, we assign a potential function for the clique whose value equals 1 for all value instantiations of the clique variables.

We use the **Numpy** library to multiplicate the *DiscreteFactor* objects and define potential functions whose values are equal to 1.

In [19]:

from pgmpy.factors.discrete import DiscreteFactor  
import numpy as np  
# clique C1: ('Z', 'X', 'Y', 'A1')  
phi\_C1 = phi\_Z  
phi\_C1.product(phi\_X)  
phi\_C1.product(phi\_Y)  
phi\_C1.product(phi\_A1)  
print(phi\_C1)  
# clique C2: ('Z', 'X', 'Y', 'E')  
phi\_C2 = DiscreteFactor(['Z', 'X', 'Y', 'E'], [2, 2, 2, 2], np.ones(16) )  
print(phi\_C2)  
# clique C3: ('Z', 'A3', 'Y', 'E')  
phi\_C3 = phi\_E  
phi\_C3.product(phi\_A3)  
print(phi\_C3)  
# clique C4: ('Z', 'X', 'D', 'E')  
phi\_C4 = DiscreteFactor(('Z', 'X', 'D', 'E'), [2, 2, 2, 2], np.ones(16))  
print(phi\_C4)  
# clique C5: ('Z', 'X', 'A2', 'D')  
phi\_C5 = phi\_D  
phi\_C5.product(phi\_A2)  
print(phi\_C5)  
# clique C6: ('A4', 'E', 'D', 'F')  
phi\_C6 = phi\_F  
phi\_C6.product(phi\_A4)  
print(phi\_C6)

+------+------+------+-------+-----------------+  
| Y | Z | X | A1 | phi(Y,Z,X,A1) |  
+======+======+======+=======+=================+  
| Y(0) | Z(0) | X(0) | A1(0) | 0.0000 |  
+------+------+------+-------+-----------------+  
| Y(0) | Z(0) | X(0) | A1(1) | 0.0063 |  
+------+------+------+-------+-----------------+  
| Y(0) | Z(0) | X(1) | A1(0) | 0.2375 |  
+------+------+------+-------+-----------------+  
| Y(0) | Z(0) | X(1) | A1(1) | 0.0063 |  
+------+------+------+-------+-----------------+  
| Y(0) | Z(1) | X(0) | A1(0) | 0.2375 |  
+------+------+------+-------+-----------------+  
| Y(0) | Z(1) | X(0) | A1(1) | 0.0063 |  
+------+------+------+-------+-----------------+  
| Y(0) | Z(1) | X(1) | A1(0) | 0.0000 |  
+------+------+------+-------+-----------------+  
| Y(0) | Z(1) | X(1) | A1(1) | 0.0063 |  
+------+------+------+-------+-----------------+  
| Y(1) | Z(0) | X(0) | A1(0) | 0.2375 |  
+------+------+------+-------+-----------------+  
| Y(1) | Z(0) | X(0) | A1(1) | 0.0063 |  
+------+------+------+-------+-----------------+  
| Y(1) | Z(0) | X(1) | A1(0) | 0.2375 |  
+------+------+------+-------+-----------------+  
| Y(1) | Z(0) | X(1) | A1(1) | 0.0063 |  
+------+------+------+-------+-----------------+  
| Y(1) | Z(1) | X(0) | A1(0) | 0.0000 |  
+------+------+------+-------+-----------------+  
| Y(1) | Z(1) | X(0) | A1(1) | 0.0063 |  
+------+------+------+-------+-----------------+  
| Y(1) | Z(1) | X(1) | A1(0) | 0.0000 |  
+------+------+------+-------+-----------------+  
| Y(1) | Z(1) | X(1) | A1(1) | 0.0063 |  
+------+------+------+-------+-----------------+  
+------+------+------+------+----------------+  
| Z | X | Y | E | phi(Z,X,Y,E) |  
+======+======+======+======+================+  
| Z(0) | X(0) | Y(0) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(0) | X(0) | Y(0) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(0) | X(0) | Y(1) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(0) | X(0) | Y(1) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(0) | X(1) | Y(0) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(0) | X(1) | Y(0) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(0) | X(1) | Y(1) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(0) | X(1) | Y(1) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(0) | Y(0) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(0) | Y(0) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(0) | Y(1) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(0) | Y(1) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(1) | Y(0) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(1) | Y(0) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(1) | Y(1) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(1) | Y(1) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
+------+------+-------+------+-----------------+  
| Y | Z | A3 | E | phi(Y,Z,A3,E) |  
+======+======+=======+======+=================+  
| Y(0) | Z(0) | A3(0) | E(0) | 0.0000 |  
+------+------+-------+------+-----------------+  
| Y(0) | Z(0) | A3(0) | E(1) | 0.9500 |  
+------+------+-------+------+-----------------+  
| Y(0) | Z(0) | A3(1) | E(0) | 0.0250 |  
+------+------+-------+------+-----------------+  
| Y(0) | Z(0) | A3(1) | E(1) | 0.0250 |  
+------+------+-------+------+-----------------+  
| Y(0) | Z(1) | A3(0) | E(0) | 0.9500 |  
+------+------+-------+------+-----------------+  
| Y(0) | Z(1) | A3(0) | E(1) | 0.0000 |  
+------+------+-------+------+-----------------+  
| Y(0) | Z(1) | A3(1) | E(0) | 0.0250 |  
+------+------+-------+------+-----------------+  
| Y(0) | Z(1) | A3(1) | E(1) | 0.0250 |  
+------+------+-------+------+-----------------+  
| Y(1) | Z(0) | A3(0) | E(0) | 0.9500 |  
+------+------+-------+------+-----------------+  
| Y(1) | Z(0) | A3(0) | E(1) | 0.0000 |  
+------+------+-------+------+-----------------+  
| Y(1) | Z(0) | A3(1) | E(0) | 0.0250 |  
+------+------+-------+------+-----------------+  
| Y(1) | Z(0) | A3(1) | E(1) | 0.0250 |  
+------+------+-------+------+-----------------+  
| Y(1) | Z(1) | A3(0) | E(0) | 0.9500 |  
+------+------+-------+------+-----------------+  
| Y(1) | Z(1) | A3(0) | E(1) | 0.0000 |  
+------+------+-------+------+-----------------+  
| Y(1) | Z(1) | A3(1) | E(0) | 0.0250 |  
+------+------+-------+------+-----------------+  
| Y(1) | Z(1) | A3(1) | E(1) | 0.0250 |  
+------+------+-------+------+-----------------+  
+------+------+------+------+----------------+  
| Z | X | D | E | phi(Z,X,D,E) |  
+======+======+======+======+================+  
| Z(0) | X(0) | D(0) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(0) | X(0) | D(0) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(0) | X(0) | D(1) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(0) | X(0) | D(1) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(0) | X(1) | D(0) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(0) | X(1) | D(0) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(0) | X(1) | D(1) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(0) | X(1) | D(1) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(0) | D(0) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(0) | D(0) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(0) | D(1) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(0) | D(1) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(1) | D(0) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(1) | D(0) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(1) | D(1) | E(0) | 1.0000 |  
+------+------+------+------+----------------+  
| Z(1) | X(1) | D(1) | E(1) | 1.0000 |  
+------+------+------+------+----------------+  
+-------+------+------+------+-----------------+  
| A2 | Z | X | D | phi(A2,Z,X,D) |  
+=======+======+======+======+=================+  
| A2(0) | Z(0) | X(0) | D(0) | 0.0000 |  
+-------+------+------+------+-----------------+  
| A2(0) | Z(0) | X(0) | D(1) | 0.9500 |  
+-------+------+------+------+-----------------+  
| A2(0) | Z(0) | X(1) | D(0) | 0.9500 |  
+-------+------+------+------+-----------------+  
| A2(0) | Z(0) | X(1) | D(1) | 0.0000 |  
+-------+------+------+------+-----------------+  
| A2(0) | Z(1) | X(0) | D(0) | 0.9500 |  
+-------+------+------+------+-----------------+  
| A2(0) | Z(1) | X(0) | D(1) | 0.0000 |  
+-------+------+------+------+-----------------+  
| A2(0) | Z(1) | X(1) | D(0) | 0.9500 |  
+-------+------+------+------+-----------------+  
| A2(0) | Z(1) | X(1) | D(1) | 0.0000 |  
+-------+------+------+------+-----------------+  
| A2(1) | Z(0) | X(0) | D(0) | 0.0250 |  
+-------+------+------+------+-----------------+  
| A2(1) | Z(0) | X(0) | D(1) | 0.0250 |  
+-------+------+------+------+-----------------+  
| A2(1) | Z(0) | X(1) | D(0) | 0.0250 |  
+-------+------+------+------+-----------------+  
| A2(1) | Z(0) | X(1) | D(1) | 0.0250 |  
+-------+------+------+------+-----------------+  
| A2(1) | Z(1) | X(0) | D(0) | 0.0250 |  
+-------+------+------+------+-----------------+  
| A2(1) | Z(1) | X(0) | D(1) | 0.0250 |  
+-------+------+------+------+-----------------+  
| A2(1) | Z(1) | X(1) | D(0) | 0.0250 |  
+-------+------+------+------+-----------------+  
| A2(1) | Z(1) | X(1) | D(1) | 0.0250 |  
+-------+------+------+------+-----------------+  
+------+------+-------+------+-----------------+  
| F | E | A4 | D | phi(F,E,A4,D) |  
+======+======+=======+======+=================+  
| F(0) | E(0) | A4(0) | D(0) | 0.0000 |  
+------+------+-------+------+-----------------+  
| F(0) | E(0) | A4(0) | D(1) | 0.9500 |  
+------+------+-------+------+-----------------+  
| F(0) | E(0) | A4(1) | D(0) | 0.0250 |  
+------+------+-------+------+-----------------+  
| F(0) | E(0) | A4(1) | D(1) | 0.0250 |  
+------+------+-------+------+-----------------+  
| F(0) | E(1) | A4(0) | D(0) | 0.9500 |  
+------+------+-------+------+-----------------+  
| F(0) | E(1) | A4(0) | D(1) | 0.9500 |  
+------+------+-------+------+-----------------+  
| F(0) | E(1) | A4(1) | D(0) | 0.0250 |  
+------+------+-------+------+-----------------+  
| F(0) | E(1) | A4(1) | D(1) | 0.0250 |  
+------+------+-------+------+-----------------+  
| F(1) | E(0) | A4(0) | D(0) | 0.9500 |  
+------+------+-------+------+-----------------+  
| F(1) | E(0) | A4(0) | D(1) | 0.0000 |  
+------+------+-------+------+-----------------+  
| F(1) | E(0) | A4(1) | D(0) | 0.0250 |  
+------+------+-------+------+-----------------+  
| F(1) | E(0) | A4(1) | D(1) | 0.0250 |  
+------+------+-------+------+-----------------+  
| F(1) | E(1) | A4(0) | D(0) | 0.0000 |  
+------+------+-------+------+-----------------+  
| F(1) | E(1) | A4(0) | D(1) | 0.0000 |  
+------+------+-------+------+-----------------+  
| F(1) | E(1) | A4(1) | D(0) | 0.0250 |  
+------+------+-------+------+-----------------+  
| F(1) | E(1) | A4(1) | D(1) | 0.0250 |  
+------+------+-------+------+-----------------+

## Quantification of the Join Tree[¶](#Quantification-of-the-Join-Tree)

The following code quantifies the join tree by adding all the potential functions of the cliques, which are the join tree factors of *Pgmpy* type *DiscreteFactor* objects.

To add the factors to the join tree object, *circuit\_JT*, we use the **add\_factors()** method of the *Pgmpy* *JunctionTree* object.

Next, we use the **check\_model()** method of the *pgmpy JunctionTree* object to verify the join tree model.

In [20]:

circuit\_JT.add\_factors(phi\_C1, phi\_C2, phi\_C3, phi\_C4, phi\_C5, phi\_C6)  
circuit\_JT.check\_model()

Out[20]:

True

## Inference by Belief Propagation[¶](#Inference-by-Belief-Propagation)

The following code imports the **BeliefProbpagation** library from the **Pgmpy.inference.ExactInference** package.

The code uses the **query()** method of the **BeliefPropagation** object

The query takes two parameters:

1. a list of variables on whom the posterior distribution is defined
2. a list of observed variable value pairs on which the posterior distribution is conditioned.

## Belief Propagation using the Pgmpy Join Tree[¶](#Xd7a35c1353acb5b0e95461e4db6b0005939efdb)

In the following, we formulate queries for inference tasks using belief propagation with the join tree

***JT***

computed by the **Pgmpy** method

### Query 1.1 Finding the Posterior Probability on assumption variables given evidence e1[¶](#Xcd300ffbbb35f791b35b9eb2381c0dc1230e1c7)

The query in the code below returns the posterior probability distribution over all four variables

['A1', 'A2', 'A3', 'A4']

conditioned on the evidence corresponding to the evidence e1 where the input variables 'X' and 'Y' being on (value 1) and the output variable 'F' being off (value 0).

{'X': 1, 'Y': 1, 'F': 0})

The posterior probability distribution determines the probability over all single and multiple faults for the circuit components.

In [21]:

from pgmpy.inference.ExactInference import BeliefPropagation  
  
belief\_propagation = BeliefPropagation(JT)  
res1 = belief\_propagation.query(  
 variables=['A1', 'A2', 'A3', 'A4'],  
 evidence={'X': 1, 'Y': 1, 'F': 0})  
print(res1)

+-------+-------+-------+-------+--------------------+  
| A1 | A2 | A3 | A4 | phi(A1,A2,A3,A4) |  
+=======+=======+=======+=======+====================+  
| A1(0) | A2(0) | A3(0) | A4(0) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(0) | A3(0) | A4(1) | 0.2981 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(0) | A3(1) | A4(0) | 0.2981 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(0) | A3(1) | A4(1) | 0.0157 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(0) | A4(0) | 0.2981 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(0) | A4(1) | 0.0157 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(1) | A4(0) | 0.0235 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(1) | A4(1) | 0.0008 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(0) | A4(0) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(0) | A4(1) | 0.0157 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(1) | A4(0) | 0.0157 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(1) | A4(1) | 0.0008 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(0) | A4(0) | 0.0157 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(0) | A4(1) | 0.0008 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(1) | A4(0) | 0.0012 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(1) | A4(1) | 0.0000 |  
+-------+-------+-------+-------+--------------------+

### Query 1.2- Finding the Posterior Probability on assumption variables given evidence e2[¶](#X4b771decaf19686f28b9bc46890bb37293e42ae)

The query in the code below finds the posterior probability distribution over all four variables,

['A1', 'A2', 'A3', 'A4']

given the evidence **e2** where the input variables 'X' and 'Y' are on (value 1), the output variable 'F' is off (value 0), and the output variable 'D' of component #2 is on (value 1)

{'X': 1, 'Y': 1, 'F': 0, 'D': 1}

The query results are expected to give us a comprehensive probability distribution over all single and multiple faults for the circuit components.

Relative to query 1, the results of query 2 due to added observation will significantly strengthen our inference, making it more specific and reliable..

In [22]:

from pgmpy.inference.ExactInference import BeliefPropagation  
  
belief\_propagation = BeliefPropagation(JT)  
res2 = belief\_propagation.query(  
 variables=['A1', 'A2', 'A3', 'A4'],  
 evidence={'X': 1, 'Y': 1, 'F': 0, 'D': 1})  
print(res2)

+-------+-------+-------+-------+--------------------+  
| A1 | A2 | A3 | A4 | phi(A1,A2,A3,A4) |  
+=======+=======+=======+=======+====================+  
| A1(0) | A2(0) | A3(0) | A4(0) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(0) | A3(0) | A4(1) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(0) | A3(1) | A4(0) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(0) | A3(1) | A4(1) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(0) | A4(0) | 0.8794 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(0) | A4(1) | 0.0231 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(1) | A4(0) | 0.0463 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(1) | A4(1) | 0.0012 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(0) | A4(0) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(0) | A4(1) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(1) | A4(0) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(1) | A4(1) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(0) | A4(0) | 0.0463 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(0) | A4(1) | 0.0012 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(1) | A4(0) | 0.0024 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(1) | A4(1) | 0.0001 |  
+-------+-------+-------+-------+--------------------+

## Query 1.3- Computing the MAP for assumption variables given evidence e1[¶](#Xa85193caa11ffd83edf0fc64c21f7bfc19f1c97)

The query in the code below is designed to compute the Maximum Posteriori Hypothesis (MAP) over all four variables,

['A1', 'A2', 'A3', 'A4'],

subject to the evidence **e1** where the input variables 'X' and 'Y' are on (value 1) and the output variable 'F' is off (value 0),

{'X': 1, 'Y': 1, 'F': 0})

The MAP query is designed to return the value assignment for the four assumption variables, which have the maximum posterior probability over all value instantiations of other unobserved variables, given the evidence **e1**.

In [23]:

from pgmpy.inference.ExactInference import BeliefPropagation  
  
belief\_propagation = BeliefPropagation(JT)  
res3 = belief\_propagation.map\_query(  
 variables=['A1', 'A2', 'A3', 'A4'],  
 evidence={'X': 1, 'Y': 1, 'F': 0})  
print(res3)

{'A1': 0, 'A2': 0, 'A3': 1, 'A4': 0}

## Query 1.4- Computing the MAP for assumption variables given evidence e2[¶](#Xb719789de114b6102ad11ca399618ba677d5889)

The query in the code below is designed to compute the Maximum Posteriori Hypothesis (MAP) over all four variables,

['A1', 'A2', 'A3', 'A4'],

given the evidence **e2** where the input variables 'X' and 'Y' are on (value 1), the output variable 'F' is off (value 0), and the output variable 'D' of component #2 is on (value 1)

{'X': 1, 'Y': 1, 'F': 0, 'D': 1}

Relative to query 3, the results of query 4 due to added observation will significantly strengthen our confidence in the computed maximum posterior hypothesis.

In [24]:

from pgmpy.inference.ExactInference import BeliefPropagation  
  
belief\_propagation = BeliefPropagation(JT)  
res4 = belief\_propagation.map\_query(  
 variables=['A1', 'A2', 'A3', 'A4'],  
 evidence={'X': 1, 'Y': 1, 'F': 0, 'D': 1})  
print(res4)

{'A1': 0, 'A2': 1, 'A3': 0, 'A4': 0}

## Query 1.5- Computing the MPE given evidence e1[¶](#X1dd5f974f940d1e384d5c142bd15c4d621c0b12)

The code below computes the most probable explanation (MPE) over the unobserved variables,

['A1', 'A2', 'A3', 'A4', 'Z', 'D', 'E']

subject to the evidence **e1** where the input variables 'X' and 'Y' are on (value 1) and the output variable 'F' is off (value 0),

{'X': 1, 'Y': 1, 'F': 0})

In [25]:

from pgmpy.inference.ExactInference import BeliefPropagation  
  
belief\_propagation = BeliefPropagation(JT)  
res5 = belief\_propagation.map\_query(  
 variables=['A1', 'A2', 'A3', 'A4', 'Z', 'D', 'E'],  
 evidence={'X': 1, 'Y': 1, 'F': 0})  
print(res5)

{'A1': 0, 'A2': 0, 'A3': 1, 'A4': 0, 'Z': 0, 'D': 0, 'E': 1}

### Query 1.6- Computing the MPE given evidence e2[¶](#X47e46c0788080dbd7daa171ba0e2640110677a9)

The code below computes the most probable explanation (MPE) over the unobserved variables,

['A1', 'A2', 'A3', 'A4', 'Z', 'E']

given the evidence **e2** where the input variables 'X' and 'Y' are on (value 1), the output variable 'F' is off (value 0), and the output variable 'D' of component #2 is on (value 1)

{'X': 1, 'Y': 1, 'F': 0, 'D': 1}

In [26]:

from pgmpy.inference.ExactInference import BeliefPropagation  
  
belief\_propagation = BeliefPropagation(JT)  
res6 = belief\_propagation.map\_query(  
 variables=['A1', 'A2', 'A3', 'A4', 'Z', 'E'],  
 evidence={'X': 1, 'Y': 1, 'F': 0, 'D': 1})  
print(res6)

{'A1': 0, 'A2': 1, 'A3': 0, 'A4': 0, 'Z': 0, 'E': 0}

## Belief Propagation Computed by the Tree Clustering Algorithm[¶](#X268e1bc83ce5953bf15ad5b2ccc3abc99918769)

In the following, we formulate queries for inference tasks using belief propagation with the join tree

***circuit\_JT***

computed by the *Tree Clustering Algorithm*.

### Query 2.1- Finding the Posterior Probability on assumption variables given evidence e1[¶](#Xae18868d1c36178e736e45fdc49a719279f5df5)

The query in the code below returns the posterior probability distribution over all four variables

['A1', 'A2', 'A3', 'A4']

conditioned on the evidence corresponding to the evidence e1 where the input variables 'X' and 'Y' being on (value 1) and the output variable 'F' being off (value 0).

{'X': 1, 'Y': 1, 'F': 0})

The posterior probability distribution determines the probability over all single and multiple faults for the circuit components.

In [27]:

from pgmpy.inference.ExactInference import BeliefPropagation  
  
belief\_propagation = BeliefPropagation(circuit\_JT)  
res1 = belief\_propagation.query(  
 variables=['A1', 'A2', 'A3', 'A4'],  
 evidence={'X': 1, 'Y': 1, 'F': 0})  
print(res1)

+-------+-------+-------+-------+--------------------+  
| A1 | A2 | A3 | A4 | phi(A1,A2,A3,A4) |  
+=======+=======+=======+=======+====================+  
| A1(0) | A2(0) | A3(0) | A4(0) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(0) | A3(0) | A4(1) | 0.2981 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(0) | A3(1) | A4(0) | 0.2981 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(0) | A3(1) | A4(1) | 0.0157 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(0) | A4(0) | 0.2981 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(0) | A4(1) | 0.0157 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(1) | A4(0) | 0.0235 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(1) | A4(1) | 0.0008 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(0) | A4(0) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(0) | A4(1) | 0.0157 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(1) | A4(0) | 0.0157 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(1) | A4(1) | 0.0008 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(0) | A4(0) | 0.0157 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(0) | A4(1) | 0.0008 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(1) | A4(0) | 0.0012 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(1) | A4(1) | 0.0000 |  
+-------+-------+-------+-------+--------------------+

### Query 2.2- Finding the Posterior Probability on assumption variables given evidence e2[¶](#Xd4a632ef01ef29d02ef5487ebc685cf8a76bd1d)

The query in the code below finds the posterior probability distribution over all four variables,

['A1', 'A2', 'A3', 'A4']

given the evidence **e2** where the input variables 'X' and 'Y' are on (value 1), the output variable 'F' is off (value 0), and the output variable 'D' of component #2 is on (value 1)

{'X': 1, 'Y': 1, 'F': 0, 'D': 1}

The query results are expected to give us a comprehensive probability distribution over all single and multiple faults for the circuit components.

Relative to query 1, the results of query 2 due to added observation will significantly strengthen our inference, making it more specific and reliable.

In [28]:

from pgmpy.inference.ExactInference import BeliefPropagation  
  
belief\_propagation = BeliefPropagation(circuit\_JT)  
res2 = belief\_propagation.query(  
 variables=['A1', 'A2', 'A3', 'A4'],  
 evidence={'X': 1, 'Y': 1, 'F': 0, 'D': 1})  
print(res2)

+-------+-------+-------+-------+--------------------+  
| A1 | A2 | A3 | A4 | phi(A1,A2,A3,A4) |  
+=======+=======+=======+=======+====================+  
| A1(0) | A2(0) | A3(0) | A4(0) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(0) | A3(0) | A4(1) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(0) | A3(1) | A4(0) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(0) | A3(1) | A4(1) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(0) | A4(0) | 0.8794 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(0) | A4(1) | 0.0231 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(1) | A4(0) | 0.0463 |  
+-------+-------+-------+-------+--------------------+  
| A1(0) | A2(1) | A3(1) | A4(1) | 0.0012 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(0) | A4(0) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(0) | A4(1) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(1) | A4(0) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(0) | A3(1) | A4(1) | 0.0000 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(0) | A4(0) | 0.0463 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(0) | A4(1) | 0.0012 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(1) | A4(0) | 0.0024 |  
+-------+-------+-------+-------+--------------------+  
| A1(1) | A2(1) | A3(1) | A4(1) | 0.0001 |  
+-------+-------+-------+-------+--------------------+

## Query 2.3- Computing the MAP for assumption variables given evidence e1[¶](#X4ae98dca74c8da74e485d4e406bcbf2c85a8b36)

The query in the code below is designed to compute the Maximum Posteriori Hypothesis (MAP) over all four variables,

['A1', 'A2', 'A3', 'A4'],

subject to the evidence **e1** where the input variables 'X' and 'Y' are on (value 1) and the output variable 'F' is off (value 0),

{'X': 1, 'Y': 1, 'F': 0})

The MAP query is designed to return the value assignment for the four assumption variables, which have the maximum posterior probability over all value instantiations of other unobserved variables, given the evidence **e1**.

In [29]:

from pgmpy.inference.ExactInference import BeliefPropagation  
  
belief\_propagation = BeliefPropagation(circuit\_JT)  
res3 = belief\_propagation.map\_query(  
 variables=['A1', 'A2', 'A3', 'A4'],  
 evidence={'X': 1, 'Y': 1, 'F': 0})  
print(res3)

{'A1': 0, 'A2': 0, 'A3': 1, 'A4': 0}

## Query 2.4- Computing the MAP for assumption variables given evidence e2[¶](#X602ba4fb074f4aae708bbfc9e261ad1a31549fe)

The query in the code below is designed to compute the Maximum Posteriori Hypothesis (MAP) over all four variables,

['A1', 'A2', 'A3', 'A4'],

given the evidence **e2** where the input variables 'X' and 'Y' are on (value 1), the output variable 'F' is off (value 0), and the output variable 'D' of component #2 is on (value 1)

{'X': 1, 'Y': 1, 'F': 0, 'D': 1}

Relative to query 3, the results of query 4 due to added observation will significantly strengthen our confidence in the computed maximum posterior hypothesis.

In [30]:

from pgmpy.inference.ExactInference import BeliefPropagation  
  
belief\_propagation = BeliefPropagation(circuit\_JT)  
res4 = belief\_propagation.map\_query(  
 variables=['A1', 'A2', 'A3', 'A4'],  
 evidence={'X': 1, 'Y': 1, 'F': 0, 'D': 1})  
print(res4)

{'A1': 0, 'A2': 1, 'A3': 0, 'A4': 0}

## Query 2.5- Computing the MPE given evidence e1[¶](#X4bf28470977707a911587bfcee99cd3bf894f8b)

The code below computes the most probable explanation (MPE) over the unobserved variables,

['A1', 'A2', 'A3', 'A4', 'Z', 'D', 'E']

subject to the evidence **e1** where the input variables 'X' and 'Y' are on (value 1) and the output variable 'F' is off (value 0),

{'X': 1, 'Y': 1, 'F': 0})

In [31]:

from pgmpy.inference.ExactInference import BeliefPropagation  
  
belief\_propagation = BeliefPropagation(circuit\_JT)  
res5 = belief\_propagation.map\_query(  
 variables=['A1', 'A2', 'A3', 'A4','Z', 'D','E'],  
 evidence={'X': 1, 'Y': 1, 'F': 0})  
print(res5)

{'A1': 0, 'A2': 0, 'A3': 0, 'A4': 1, 'Z': 0, 'D': 0, 'E': 0}

### Query 2.6- Computing the MPE given evidence e2[¶](#Xae65554b848c84847731a72a409a40724a795fb)

The code below computes the most probable explanation (MPE) over the unobserved variables,

['A1', 'A2', 'A3', 'A4', 'Z', 'E']

given the evidence **e2** where the input variables 'X' and 'Y' are on (value 1), the output variable 'F' is off (value 0), and the output variable 'D' of component #2 is on (value 1)

{'X': 1, 'Y': 1, 'F': 0, 'D': 1}

In [32]:

from pgmpy.inference.ExactInference import BeliefPropagation  
  
belief\_propagation = BeliefPropagation(circuit\_JT)  
res6 = belief\_propagation.map\_query(  
 variables=['A1', 'A2', 'A3', 'A4', 'Z', 'E'],  
 evidence={'X': 1, 'Y': 1, 'F': 0, 'D': 1})  
print(res6)

{'A1': 0, 'A2': 1, 'A3': 0, 'A4': 0, 'Z': 0, 'E': 0}