

# Variable Elimination

---

December 4, 2018

## Contents

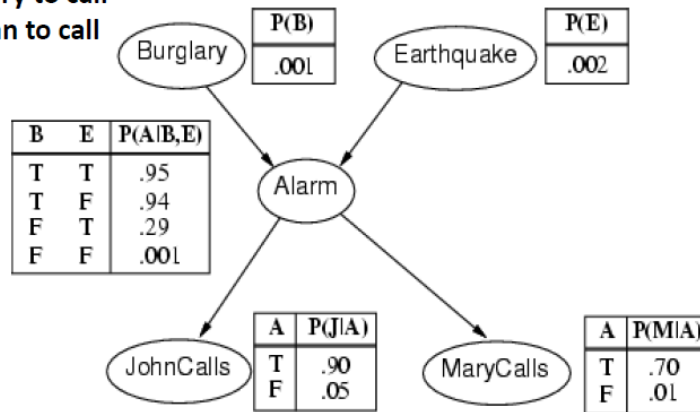
<b>1</b>	<b>VE</b>	<b>2</b>
<b>2</b>	<b>Task</b>	<b>5</b>
<b>3</b>	<b>Codes and Results</b>	<b>5</b>

# 1 VE

The burglary example is described as following:

- A burglary can set the alarm off
- An earthquake can set the alarm off
- The alarm can cause Mary to call
- The alarm can cause John to call

Note that these tables only provide the probability that  $X_i$  is true.  
(E.g.,  $\Pr(A \text{ is true} | B, E)$ )  
The probability that  $X_i$  is false is 1- these values



```
P(Alarm) =
0.002516442

P(J&&~M) =
0.050054875461

P(A | J&&~M) =
0.0135738893313

P(B | A) =
0.373551228282

P(B | J&&~M) =
0.0051298581334

P(J&&~M | ~B) =
0.049847949
```

Here is a VE template for you to solve the burglary example:

```
class VariableElimination:
    @staticmethod
    def inference(factorList, queryVariables,
        orderedListOfHiddenVariables, evidenceList):
        for ev in evidenceList:
            #Your code here
        for var in orderedListOfHiddenVariables:
            #Your code here
        print "RESULT:"
```

```

        res = factorList[0]
        for factor in factorList[1:]:
            res = res.multiply(factor)
        total = sum(res.cpt.values())
        res.cpt = {k: v/total for k, v in res.cpt.items()}
        res.printInf()

    @staticmethod
    def printFactors(factorList):
        for factor in factorList:
            factor.printInf()

class Util:
    @staticmethod
    def to_binary(num, len):
        return format(num, '0' + str(len) + 'b')

class Node:
    def __init__(self, name, var_list):
        self.name = name
        self.varList = var_list
        self.cpt = {}

    def setCpt(self, cpt):
        self.cpt = cpt

    def printInf(self):
        print "Name_=" + self.name
        print "_vars_" + str(self.varList)
        for key in self.cpt:
            print "___key:_ " + key + "_val:_ " + str(self.cpt[key])
        print ""

    def multiply(self, factor):
        """function that multiplies with another factor"""
        #Your code here
        new_node = Node("f" + str(newList), newList)
        new_node.setCpt(new_cpt)
        return new_node

```

```

def sumout(self, variable):
    """function that sums out a variable given a factor"""
    #Your code here
    new_node = Node("f" + str(new_var_list), new_var_list)
    new_node.setCpt(new_cpt)
    return new_node

def restrict(self, variable, value):
    """function that restricts a variable to some value
    in a given factor"""
    #Your code here
    new_node = Node("f" + str(new_var_list), new_var_list)
    new_node.setCpt(new_cpt)
    return new_node

# create nodes for Bayes Net
B = Node("B", ["B"])
E = Node("E", ["E"])
A = Node("A", ["A", "B", "E"])
J = Node("J", ["J", "A"])
M = Node("M", ["M", "A"])

# Generate cpt for each node
B.setCpt({'0': 0.999, '1': 0.001})
E.setCpt({'0': 0.998, '1': 0.002})
A.setCpt({'111': 0.95, '011': 0.05, '110':0.94, '010':0.06,
'101':0.29, '001':0.71, '100':0.001, '000':0.999})
J.setCpt({'11': 0.9, '01': 0.1, '10': 0.05, '00': 0.95})
M.setCpt({'11': 0.7, '01': 0.3, '10': 0.01, '00': 0.99})

print "P(A) ⊥*****"
VariableElimination.inference([B,E,A,J,M], ['A'], ['B', 'E', 'J', 'M'], {})

print "P(B ⊥ J ⊥ M) ⊥*****"
VariableElimination.inference([B,E,A,J,M], ['B'], ['E', 'A'], {'J':1, 'M':0})

```

## 2 Task

- You should implement 4 functions: `inference`, `multiply`, `sumout` and `restrict`. You can turn to Figure 1 and Figure 2 for help.

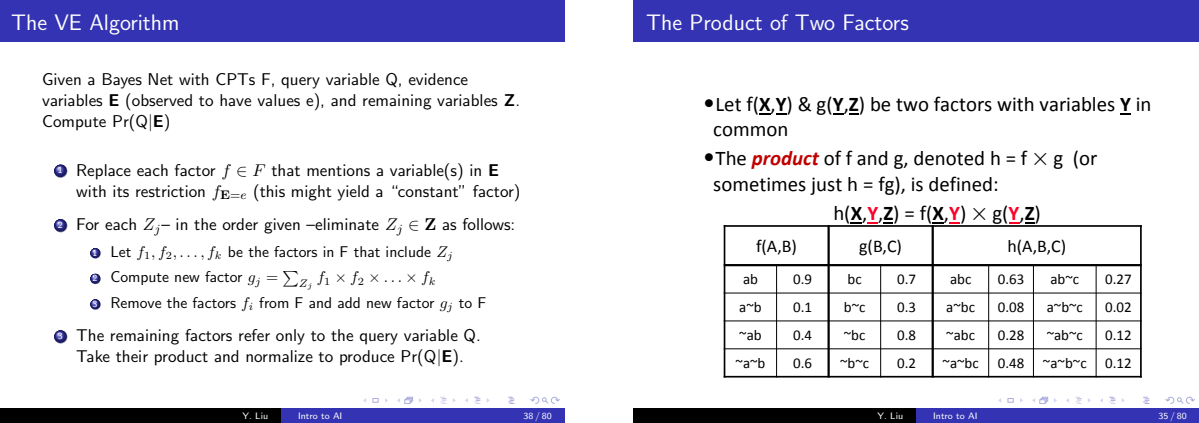


Figure 1: VE and Product

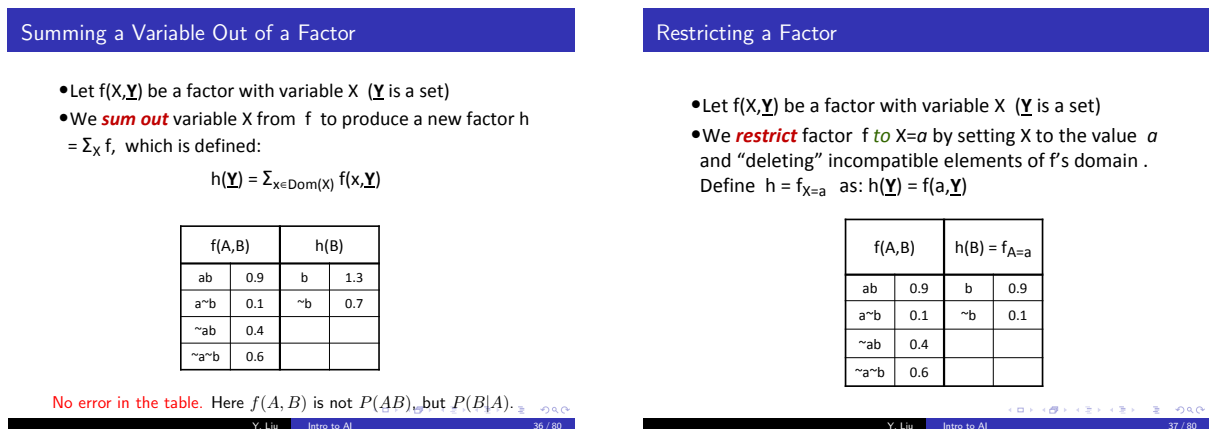


Figure 2: Sumout and Restrict

## 3 Codes and Results