School of Data and Computer Science, Sun Yat-sen University

E09 Variable Elimination

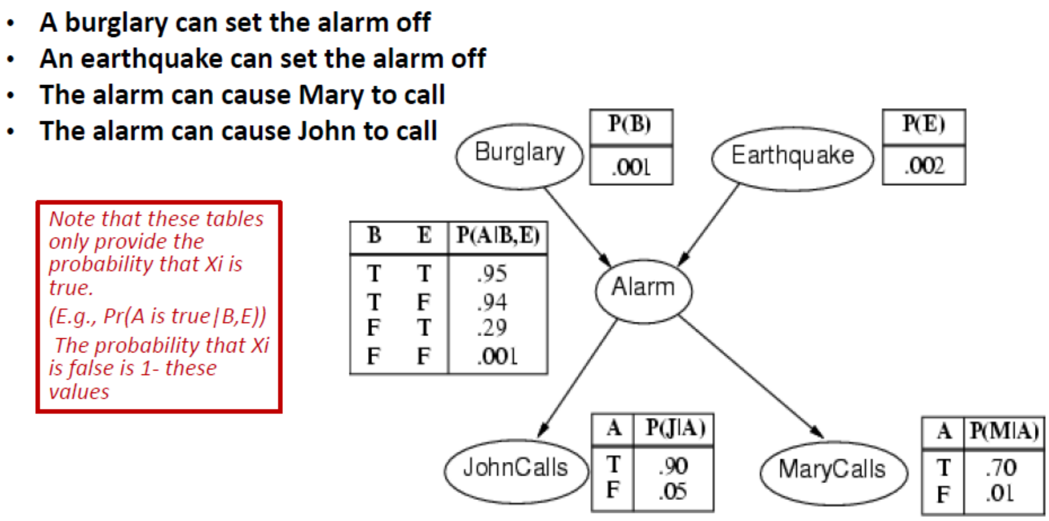
18364066 Yanzuo Lu

November 10, 2020

Contents

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 Xi is true.

(E.g., Pr(A is true|B,E))

The probability that Xi is false is 1- these values

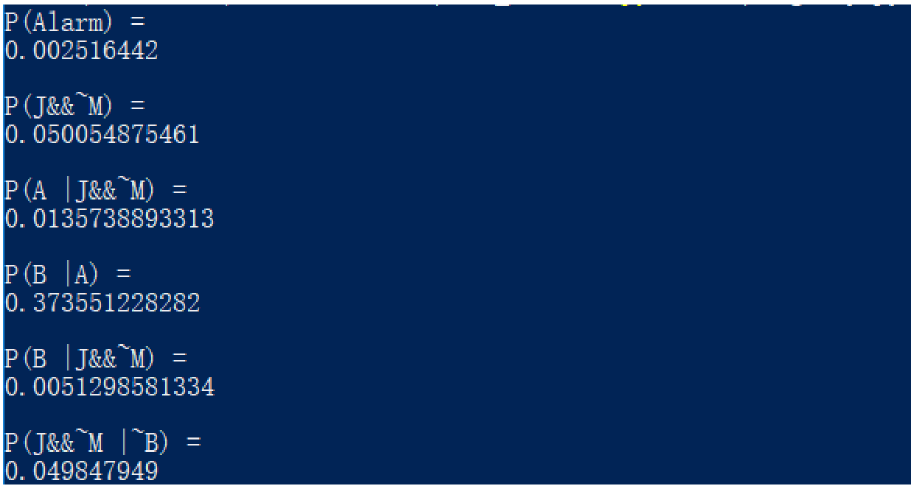
P(B)
.001

P(E)
.002

B	E	P(A|B,E)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

A	P(J|A)
T	.90
F	.05

A	P(M|A)
T	.70
F	.01



```
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
```

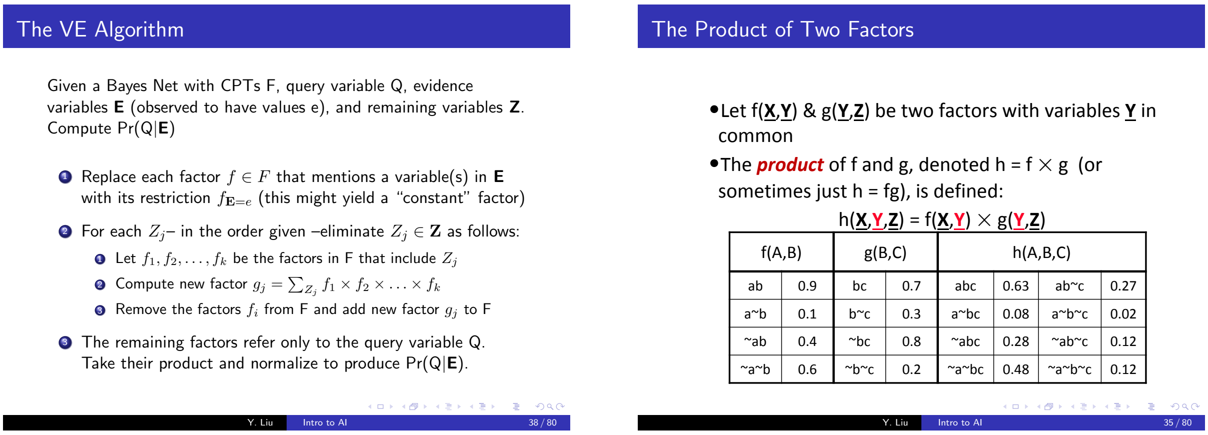
Here is a VE template for you to solve the burglary example:

```
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.
- Please hand in a file named E09_YourNumber.pdf, and send it to ai_2020@foxmail.com



The VE Algorithm

Given a Bayes Net with CPTs F, query variable Q, evidence variables **E** (observed to have values e), and remaining variables **Z**. Compute Pr(Q|**E**)

1. Replace each factor $f \in F$ that mentions a variable(s) in **E** with its restriction $f_{\mathbf{E}=e}$ (this might yield a "constant" factor)
2. For each Z_j– in the order given –eliminate $Z_j \in \mathbf{Z}$ as follows:
   1. Let $f_1, f_2, \ldots, f_k$ be the factors in F that include Z_j
   2. Compute new factor $g_j = \sum_{Z_j} f_1 \times f_2 \times \ldots \times f_k$
   3. Remove the factors f_i from F and add new factor g_j to F
3. The remaining factors refer only to the query variable Q. Take their product and normalize to produce Pr(Q|**E**).

Y. Liu Intro to AI 38 / 80

The Product of Two Factors

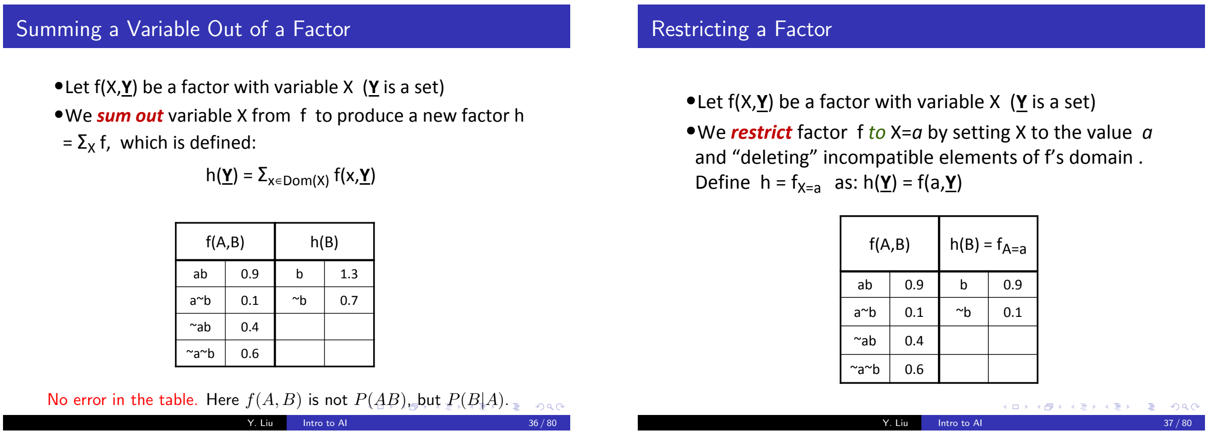
- Let f(**X**,**Y**) & g(**Y**,**Z**) be two factors with variables **Y** in common
- The ***product*** of f and g, denoted h = f × g (or sometimes just h = fg), is defined:

h(**X**,**Y**,**Z**) = f(**X**,**Y**) × g(**Y**,**Z**)

f(A,B)		g(B,C)		h(A,B,C)			
ab	0.9	bc	0.7	abc	0.63	ab~c	0.27
a~b	0.1	b~c	0.3	a~bc	0.08	a~b~c	0.02
~ab	0.4	~bc	0.8	~abc	0.28	~ab~c	0.12
~a~b	0.6	~b~c	0.2	~a~bc	0.48	~a~b~c	0.12

Y. Liu Intro to AI 35 / 80

Figure 1: VE and Product



Summing a Variable Out of a Factor

- Let f(X,**Y**) be a factor with variable X (**Y** is a set)
- We ***sum out*** variable X from f to produce a new factor h = Σ_X f, which is defined:

h(**Y**) = $\Sigma_{x \in Dom(X)}$ f(x,**Y**)

f(A,B)		h(B)	
ab	0.9	b	1.3
a~b	0.1	~b	0.7
~ab	0.4		
~a~b	0.6		

No error in the table. Here $f(A,B)$ is not $P(AB)$, but $P(B|A)$.

Y. Liu Intro to AI 36 / 80

Restricting a Factor

- Let f(X,**Y**) be a factor with variable X (**Y** is a set)
- We ***restrict*** factor f *to* X=*a* by setting X to the value *a* and "deleting" incompatible elements of f's domain . Define h = $f_{X=a}$ as: h(**Y**) = f(a,**Y**)

f(A,B)		h(B) = $f_{A=a}$	
ab	0.9	b	0.9
a~b	0.1	~b	0.1
~ab	0.4		
~a~b	0.6		

Y. Liu Intro to AI 37 / 80

Figure 2: Sumout and Restrict

3 Codes and Results

”main.py”

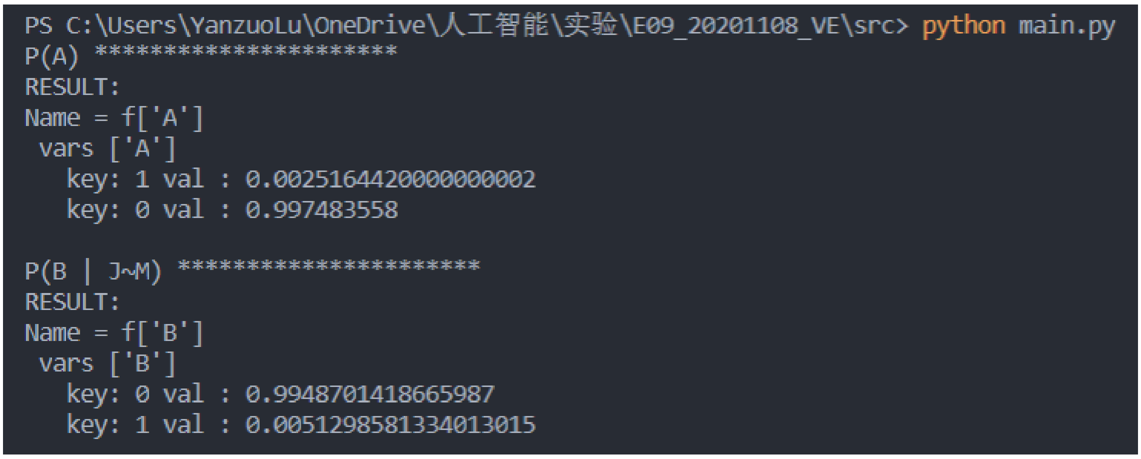
```python
from math import pow

class VariableElimination:
    @staticmethod
    def inference(factorList, queryVariables, orderedListOfHiddenVariables, evidenceList):
        for ev in evidenceList:
            #Your code here
            for i, factor in enumerate(factorList):
                if ev not in factor.varList:
                    continue
                factorList[i] = factor.restrict(ev, evidenceList[ev])
        for var in orderedListOfHiddenVariables:
            #Your code here
            index_list = []
            for i, factor in enumerate(factorList):
                if var in factor.varList:
                    index_list.append(i)
            new_factor_var = []
            for i in index_list:
                for factor_var in factorList[i].varList:
                    if factor_var not in new_factor_var and factor_var != var:
                        new_factor_var.append(factor_var)

            cal_factor = Node(”tmp”, factorList[index_list[0]].varList.copy())
            cal_factor.setCpt(factorList[index_list[0]].cpt.copy())
            for i in index_list[1:]:
                temp_factor = Node(”tmp”, factorList[i].varList.copy())
                temp_factor.setCpt(factorList[i].cpt.copy())
                cal_factor = cal_factor.multiply(temp_factor)

            new_factor = cal_factor.sumout(var)
            factorList.append(new_factor)
            factorList = [factor for i, factor in enumerate(factorList) if i not in index_list]

        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
        newList = self.varList.copy()
        not_same_index = []
        same_index = []
        for i, var in enumerate(factor.varList):
            if var not in newList:
                not_same_index.append(i)
                newList.append(var)
            else:
                same_index.append((newList.index(var), i))

        new_cpt = {}
        for cpt1 in self.cpt:
            for cpt2 in factor.cpt:
                valid_flag = True
                for si in same_index:
                    if cpt1[si[0]] != cpt2[si[1]]:
                        valid_flag = False
                        break
                if not valid_flag:
                    continue

                new_cpt_key = cpt1
                for nsi in not_same_index:
                    new_cpt_key = new_cpt_key + cpt2[nsi]
                if new_cpt_key in new_cpt:
                    new_cpt[new_cpt_key] += self.cpt[cpt1] * factor.cpt[cpt2]
                else:
                    new_cpt[new_cpt_key] = self.cpt[cpt1] * factor.cpt[cpt2]

        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
        var_index = self.varList.index(variable)
        new_var_list = self.varList.copy()
        new_var_list.remove(variable)

        new_cpt = {}
        for cpt1 in self.cpt:
            new_cpt_key = ””
            for i, ch in enumerate(cpt1):
                if i != var_index:
                    new_cpt_key = new_cpt_key + ch

            if new_cpt_key in new_cpt:
                new_cpt[new_cpt_key] += self.cpt[cpt1]
            else:
                new_cpt[new_cpt_key] = self.cpt[cpt1]

        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
        # print(self.varList, self.cpt)
        var_index = self.varList.index(variable)
        new_var_list = self.varList.copy()
        new_var_list.remove(variable)

        new_cpt = {}
        for cpt1 in self.cpt:
            if cpt1[var_index] != str(value):
                continue

            new_cpt_key = ””
            for i, ch in enumerate(cpt1):
                if i != var_index:
                    new_cpt_key = new_cpt_key + ch

            new_cpt[new_cpt_key] = self.cpt[cpt1]

        # print(new_var_list, new_cpt)
        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})
```

本次实验要求我们实现在 uncertainty 中提到的变量消除 VariableElimination 算法，用该算法解决贝叶斯网络中最基本的问题，给定一些已知的 Evidence 变量求 Query 变量的后验概率，而变量消除的算法核心就在于 Product 和 Sum 两个操作，操作具体在 PPT 中已经阐释地较为清楚。

在实现过程中，因子 factor 之间的三种操作 Product、Sum 和 Restrict 本质上是类似的，都是基于当前 factor 的变量列表 varlist 和条件概率表 cpt 计算一个新的 factor 的 varlist 和 cpt，计算新的 varlist 在 Product 中需要剔除掉重复的变量，在 Sum 和 Restrict 中则需要删掉给定的变量，计算新的 cpt 在 Product 中需要两两相乘并把相同的进行相加，在 Sum 中则不需要乘法直接将相同的相加即可，Restrict 中则在相加的基础上去除掉不符合给定 value 的条目。

在变量消除过程中依据这三种操作，注意维护 factorlist 的增删即可，每次消除一个变量都需要删除一些 factor 并把新的 factor 加进列表中，下面是运行结果。



```
PS C:\Users\YanzuoLu\OneDrive\人工智能\实验\E09_20201108_VE\src> python main.py
P(A) **********************
RESULT:
Name = f['A']
 vars ['A']
   key: 1 val : 0.0025164420000000002
   key: 0 val : 0.997483558

P(B | J~M) **********************
RESULT:
Name = f['B']
 vars ['B']
   key: 0 val : 0.9948701418665987
   key: 1 val : 0.0051298581334013015
```

Figure 3: result.png