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

E09 Variable Elimination

18340052

November 25, 2020

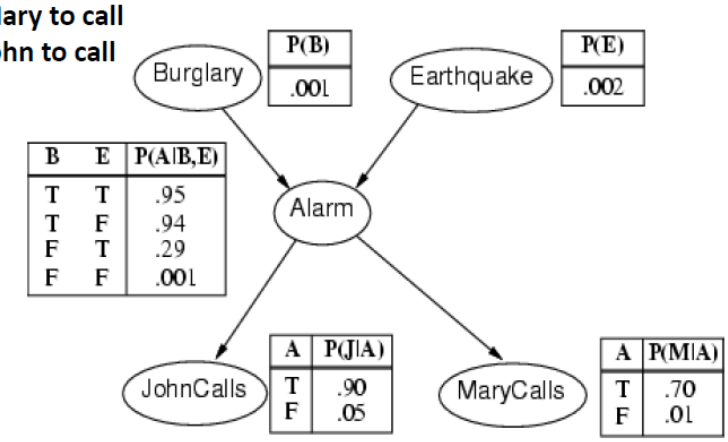
Contents

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(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 | JM) **********"
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

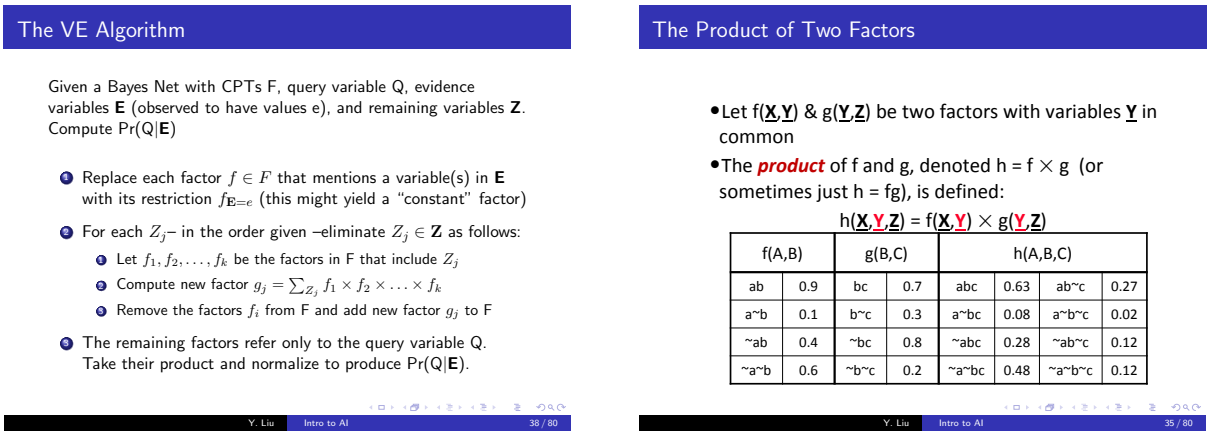


Figure 1: VE and Product

3 Codes and Results

Code:

```
class VariableElimination:
    @staticmethod
    def inference(factorList, queryVariables,
                  orderdListOfHiddenVariables, evidenceList):
        for ev in evidenceList:
            for factor in factorList:
                if ev in factor.varList:
                    factorList.append(factor.restrict(ev, evidenceList[
                        ev]))
```

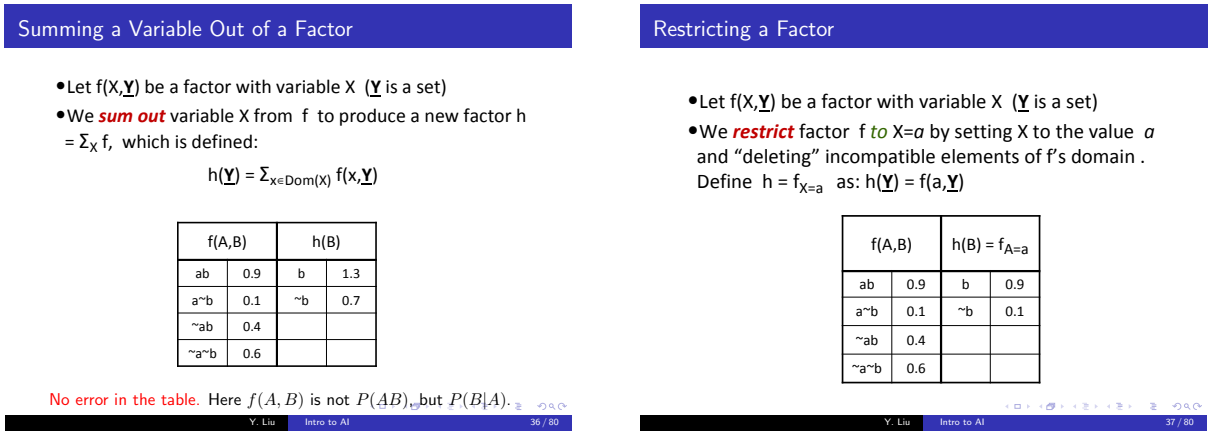


Figure 2: Sumout and Restrict

```
                    factorList.remove(factor)

        for var in orderdListOfHiddenVariables:
            eliminationList = list(filter(lambda factor: var in factor.
                varList,factorList))
            new_var = eliminationList[0]
            for e in eliminationList:
                for i in factorList:
                    if i.name == e.name:
                        factorList.remove(i)
                if not e == eliminationList[0]:
                    new_var = new_var.multiply(e)
            new_var = new_var.sumout(var)
            factorList.append(new_var)
        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')
```

```
42 class Node:
43     def __init__(self, name, var_list):
44         self.name = name
45         self.varList = var_list
46         self.cpt = {}
47 
48     def setCpt(self, cpt):
49         self.cpt = cpt
50 
51     def printInf(self):
52         print("Name = " + self.name)
53         print(" vars " + str(self.varList))
54         for key in self.cpt:
55             print("   key: " + key + " val : " + str(self.cpt[key]))
56         print()
57 
58     def multiply(self, factor):
59         newList = [var for var in self.varList]
60         new_cpt = {}
61 
62         idx1 = []
63         idx2 = []
64         for var1 in self.varList:
65             for var2 in factor.varList:
66                 if var1 == var2:
67                     idx1.append(self.varList.index(var1))
68                     idx2.append(factor.varList.index(var2))
69                 else:
70                     newList.append(var2)
71 
72         for k1, v1 in self.cpt.items():
73             for k2, v2 in factor.cpt.items():
74                 flag = True
75                 for i in range(len(idx1)):
76                     if k1[idx1[i]] != k2[idx2[i]]:
77                         flag = False
78                         break
79                 if flag:
80                     new_key = k1
81                     for i in range(len(k2)):
82                         if i in idx2: continue
83                         new_key += k2[i]
84                     new_cpt[new_key] = v1 * v2
85         new_node = Node("f" + str(newList), newList)
86         new_node.setCpt(new_cpt)
87         return new_node
88 
89     def sumout(self, variable):
90         new_var_list = [var for var in self.varList]
```

```
91          new_var_list.remove(variable)
92          new_cpt = {}
93          idx = self.varList.index(variable)
94          for k, v in self.cpt.items():
95              if k[:idx] + k[idx+1:] not in new_cpt.keys():
96                  new_cpt[k[:idx] + k[idx+1:]] = v
97              else: new_cpt[k[:idx] + k[idx+1:]] += v
98          new_node = Node("f" + str(new_var_list), new_var_list)
99          new_node.setCpt(new_cpt)
100         return new_node
101
102     def restrict(self, variable, value):
103         new_var_list = [i for i in self.varList]
104         new_var_list.remove(variable)
105         new_cpt = {}
106         idx = self.varList.index(variable)
107         for k, v in self.cpt.items():
108             if k[idx] == str(value):
109                 new_cpt[k[:idx] + k[idx+1:]] = v
110         new_node = Node("f" + str(new_var_list), new_var_list)
111         new_node.setCpt(new_cpt)
112         return new_node
113
114 B = Node("B", ["B"])
115 E = Node("E", ["E"])
116 A = Node("A", ["A", "B", "E"])
117 J = Node("J", ["J", "A"])
118 M = Node("M", ["M", "A"])
119
120 B.setCpt({'0': 0.999, '1': 0.001})
121 E.setCpt({'0': 0.998, '1': 0.002})
122 A.setCpt({'111': 0.95, '011': 0.05,'110': 0.94, '010': 0.06,
123           '101': 0.29, '001': 0.71,'100': 0.001, '000': 0.999})
124 J.setCpt({'11': 0.9, '01': 0.1,'10': 0.05, '00': 0.95})
125 M.setCpt({'11': 0.7, '01': 0.3,'10': 0.01, '00': 0.99})
126
127 print("P(A) **********")
128 VariableElimination.inference(
129         [B, E, A, J, M], ['A'],
130         ['B', 'E', 'J', 'M'],{})
131
132 print("P(J && M) **********")
133 VariableElimination.inference(
134         [B, E, A, J, M], ['J'],
135         ['B', 'E', 'A'],{'M': 0})
136
137 print("P(A | J && M) **********")
138 VariableElimination.inference(
139         [B, E, A, J, M], ['A'],
```

```
        ['B', 'E'],{'J': 1, 'M': 0})

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

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

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

Result:


..9_20201108_VE (-zsh)
⌥⌘2
heze@HeZes-MBP …/大三上/人工智能实验/E09_20201108_VE main
python3 VE.py
19:08:34
P(A) **********
RESULT:
Name = f['A']
vars ['A']
key: 1 val : 0.0025164420000000002
key: 0 val : 0.997483558
P(J && ~M) **********
RESULT:
Name = f['J', 'A', 'A']
vars ['J', 'A', 'A']
key: 1 val : 0.05064931327459069
key: 0 val : 0.9493506867254093
P(A | J && ~M) **********
RESULT:
Name = f['A']
vars ['A']
key: 1 val : 0.013573889331307633
key: 0 val : 0.9864261106686925
P(B | A) **********
RESULT:
Name = f['B', 'A']
vars ['B', 'A']
key: 01 val : 0.313224385859082
key: 00 val : 0.313224385859082
key: 11 val : 0.186775614140918
key: 10 val : 0.186775614140918
P(B | J && ~M) **********
RESULT:
Name = f['B']
vars ['B']
key: 0 val : 0.9948701418665987
key: 1 val : 0.0051298581334013015
P(J && ~M | ~B) **********
RESULT:
Name = f[]
vars []
key: 11 val : 0.001493351
key: 10 val : 0.049847948999999996
key: 01 val : 0.009595469
key: 00 val : 0.939063231
heze@HeZes-MBP …/大三上/人工智能实验/E09_20201108_VE main
19:09:07