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

P03 Planning and Uncertainty

18364066 Lu Yanzuo

November 25, 2020

Contents

1 STRIPS planner

In this part, you will implement a simple STRIPS planner. The input of your planner is a PDDL domain file and a problem file in the STRIPS restriction, that is, preconditions of actions and the goal are conjunctions of atoms, and effects of actions are conjunctions of literals. The output of your planner is a sequence of actions to achieve the goal.

1. Describe with sentences the main ideas behind computing the heuristic for a state using reachability analysis from lecture notes. (10 points)
2. Implement a STRIPS planner by using A* search and the heuristic function you implemented.(20 points)
3. Explain any ideas you use to speed up the implementation. (10 points)
4. Run you planner on the 5 test cases, and report the returned plans and the running times. Analyse the experimental results. (10 points)

2 Diagnosing by Bayesian Networks

2.1 Variables and their domais

```
(1)PatientAge:['0-30','31-65','65+']
(2)CTScanResult:['Ischemic Stroke','Hemmorraghic Stroke']
(3)MRIScanResult: ['Ischemic Stroke','Hemmorraghic Stroke']
(4)StrokeType: ['Ischemic Stroke','Hemmorraghic Stroke', 'Stroke Mimic']
(5)Anticoagulants: ['Used','Not used']
(6)Mortality:['True', 'False']
(7)Disability: ['Negligible', 'Moderate', 'Severe']
```

2.2 CPTs

[CTScanResult, MRIScanResult,StrokeType] means:

P(StrokeType='...' | CTScanResult='...' ∧ MRIScanResult='...')

```
(1)
[PatientAge]


['0-30', 0.10],
['31-65', 0.30],
['65+', 0.60]
```

(2)
[CTScanResult]

['Ischemic Stroke',0.7],
['Hemmorraghic Stroke',0.3]

(3)
[MRIScanResult]

['Ischemic Stroke',0.7],
['Hemmorraghic Stroke',0.3]

(4)
[Anticoagulants]

[Used',0.5],
['Not used',0.5]

(5)
[CTScanResult, MRIScanResult,StrokeType])

['Ischemic Stroke','Ischemic Stroke','Ischemic Stroke',0.8],
['Ischemic Stroke','Hemmorraghic Stroke','Ischemic Stroke',0.5],
['Hemmorraghic Stroke','Ischemic Stroke','Ischemic Stroke',0.5],
['Hemmorraghic Stroke','Hemmorraghic Stroke','Ischemic Stroke',0],

['Ischemic Stroke','Ischemic Stroke','Hemmorraghic Stroke',0],
['Ischemic Stroke','Hemmorraghic Stroke','Hemmorraghic Stroke',0.4],
['Hemmorraghic Stroke','Ischemic Stroke','Hemmorraghic Stroke',0.4],
['Hemmorraghic Stroke','Hemmorraghic Stroke','Hemmorraghic Stroke
',0.9],

```
['Ischemic Stroke','Ischemic Stroke','Stroke Mimic',0.2],
['Ischemic Stroke','Hemmorraghic Stroke','Stroke Mimic',0.1],
[ 'Hemmorraghic Stroke','Ischemic Stroke','Stroke Mimic',0.1],
[ 'Hemmorraghic Stroke','Hemmorraghic Stroke','Stroke Mimic',0.1],

(6)
[StrokeType, Anticoagulants, Mortality]

['Ischemic Stroke', 'Used', 'False',0.28],
['Hemmorraghic Stroke', 'Used', 'False',0.99],
['Stroke Mimic', 'Used', 'False',0.1],
['Ischemic Stroke','Not used', 'False',0.56],
['Hemmorraghic Stroke', 'Not used', 'False',0.58],
['Stroke Mimic', 'Not used', 'False',0.05],

['Ischemic Stroke', 'Used' ,'True',0.72],
['Hemmorraghic Stroke', 'Used', 'True',0.01],
['Stroke Mimic', 'Used', 'True',0.9],
['Ischemic Stroke', 'Not used' ,'True',0.44],
['Hemmorraghic Stroke', 'Not used', 'True',0.42 ],
['Stroke Mimic', 'Not used', 'True',0.95]

(7)
[StrokeType, PatientAge, Disability]

['Ischemic Stroke',   '0-30','Negligible', 0.80],
['Hemmorraghic Stroke', '0-30','Negligible', 0.70],
['Stroke Mimic',        '0-30', 'Negligible',0.9],
['Ischemic Stroke',     '31-65','Negligible', 0.60],
['Hemmorraghic Stroke', '31-65','Negligible', 0.50],
['Stroke Mimic',        '31-65', 'Negligible',0.4],
['Ischemic Stroke',     '65+'  , 'Negligible',0.30],
['Hemmorraghic Stroke', '65+'  , 'Negligible',0.20],
```

```
['Stroke Mimic',         '65+'   , 'Negligible',0.1],

['Ischemic Stroke',      '0-30'  ,'Moderate',0.1],
['Hemmorraghic Stroke', '0-30'  ,'Moderate',0.2],
['Stroke Mimic',         '0-30'  ,'Moderate',0.05],
['Ischemic Stroke',      '31-65','Moderate',0.3],
['Hemmorraghic Stroke', '31-65','Moderate',0.4],
['Stroke Mimic',         '31-65','Moderate',0.3],
['Ischemic Stroke',      '65+'   ,'Moderate',0.4],
['Hemmorraghic Stroke', '65+'   ,'Moderate',0.2],
['Stroke Mimic',         '65+'   ,'Moderate',0.1],

['Ischemic Stroke',      '0-30'  ,'Severe',0.1],
['Hemmorraghic Stroke', '0-30'  ,'Severe',0.1],
['Stroke Mimic',         '0-30'  ,'Severe',0.05],
['Ischemic Stroke',      '31-65','Severe',0.1],
['Hemmorraghic Stroke', '31-65','Severe',0.1],
['Stroke Mimic',         '31-65','Severe',0.3],
['Ischemic Stroke',      '65+'   ,'Severe',0.3],
['Hemmorraghic Stroke', '65+'   ,'Severe',0.6],
['Stroke Mimic',         '65+'   ,'Severe',0.8]
```

2.3 Tasks

1. Briefly describe with sentences the main ideas of the VE algorithm. (10 points)
2. Implement the VE algorithm (C++ or Python) to calculate the following probability values: (10 points)
   (a) p1 = P(Mortality='True' ∧ CTScanResult='Ischemic Stroke' | PatientAge='31-65')
   (b) p2 = P(Disability='Moderate' ∧ CTScanResult='Hemmorraghic Stroke' | PatientAge='65+' ∧ MRIScanResult='Hemmorraghic Stroke')
   (c) p3 = P(StrokeType='Hemmorraghic Stroke' | PatientAge='65+' ∧ CTScanResult='Hemmorraghic Stroke' ∧ MRIScanResult='Ischemic Stroke')
   (d) p4 = P(Anticoagulants='Used' | PatientAge='31-65')
   (e) p5 = P(Disability='Negligible')

3. Implement an algorithm to select a good order of variable elimination. (10 points)
4. Compare the running times of the VE algorithm for different orders of variable elimination, and fill out the following table: For test cases p4 and p5, for each of the order selected by your algorithm and 5 other orders, report the elimination with, and the total running time of the VE algorithm. For each case, the first order of elimination should be the one chosen by your algorithm. Analyze the results. (20 points)

Test case	Elimination order	Elimination width	Total time
p4			
p4			
p4			
p4			
p4			
p4			
p5			
p5			
p5			
p5			
p5			
p5			

3 Due: 11:59pm, Saturday, Nov. 28, 2020

Please hand in a file named P03_YourNumber.pdf, and send it to ai_2020@foxmail.com

4 Result

4.1 STRIPS planner

4.1.1 Reachability Analysis

给定一个封闭世界的数据库 CW-KB 作为初始状态，以及一组 STRIPS 操作符将一个状态映射到另一个状态，为了达到我们想要的最终目标状态，规划问题就是要找出按照特定顺序的一组操作，来让初始状态能够一步接一步最终映射到最终目标状态。而为了找出这样的一个特定顺序的一组操作，我们最常用的方法就是启发式搜索，包括之前在课内提到的 A* 算法和它的一些变种方法，但在这个规划问题中，启发式函数的值并不再是先前的问题中可以用距离来衡量的情况了，为此我们考虑该问题的松弛问题 Relaxed Problem，忽略掉 STRIPS 操作符中删除列表 delete 中的行为。

在课堂上我们已经证明出一个结论，即松弛问题的最优解长度以原问题的最优解长度为上界，这也就证明了松弛问题的解长度可以用作 A* 算法中的一个可采纳的启发式函数值。但是求解一个松弛问题的最优解也是十分困难的，属于一个 NP 难问题，为此我们可以从初始状态到最终目标状态构建分层结构，计算在松弛问题规划中需要完成多少步操作，以这个步数作为启发式函数值的估计。

下面我用若干句子简单地总结了这个分层结构是如何运作的：

1. 每一层都属于状态层 state layer 或操作层 action layer 二者其一且两层交替出现，第一层属于初始状态层 S_0
2. 若当前层是状态层 state layer，找出所有可执行的操作构成下一个操作层 action layer
3. 若当前层是操作层 action layer，下一个状态层 state layer 由上一个状态层 state layer 和当前操作层 action layer 的 adds 列表中的所有事实 facts 并在一起构成
4. 持续上述两步操作直至以下两种情况出现：一是最终目标状态已经是当前状态层 state layer 的子集，二是当前状态层 state layer 相较于上一个状态层 state layer 已经没有任何改动

若当前状态层 state layer 已经包含了最终目标状态，下面就可以开始计算我们所需要的启发式函数值，计算这个值的函数一般叫做 CountActions，这是一个递归函数，参数包括目标状态 G 和状态层 S_K，以下是该函数的流程简述：

1. 若 K=0 表示递归结束，直接返回 0 即可
2. 将目标状态 G 切分成 $G_P = G \cap S_{K-1}$ 和 $G_N = G - G_P$，这样 G_P 就包含上一个状态层已经实现的部分，G_N 就包含只在当前状态层才实现的部分
3. 随后找出 adds 列表能够包含 G_N 的一个最小操作子集 A，注意不能包含冗余操作
4. 递归调用 CountActions，递归调用的参数目标状态 G 变为 $G_P \cup preconditions\ of\ A$，状态层变为上一个状态层
5. 得到递归调用的返回值后，加上最小操作子集 A 的大小返回即可

4.1.2 Implementation

STRIPS 规划器的实现主要分为几个部分：problem pddl 解析，domain pddl 解析, heuristic 启发式函数和 Astar 算法，四个部分均由本人独立实现，从未参考或抄袭任何其他人代码。

在 problem pddl 解析中，我们需要 objects, init, goal 三个小块的内容，其中 init 初始状态和 goal 目标状态都是由若干谓词 predicate 构成，objects 由若干种类的多个物体构成，经过对样例的 problem 文件进行一定的修改，例如每一行只保留一个种类的 objects，init 中每一行只包含一个谓词，goal 所有谓词位于同一行等，利用 python 的正则表达式库 re 就可以很便捷地取出各个谓词和 objects。

同理，对于 domain pddl 解析也是这样，也需要对所有样例的 domain 文件进行一定的格式化，使得正则表达式能够正确取出相应的 parameters, precondition 和 effect。此外，由于 STRIPS 语法中所有动作 action 的先决条件 precondition 都是必须为真的，在对 domain 文件的格式化中需要添加一些谓词

（这只是一种解决方案，可能不需要添加新的谓词也能实现，但直接添加新的谓词需要保证语义不发生变化），例如在样例 0 和样例 1 中，我添加了 not_guarded 表示该 location 没有被怪物守卫，相应地在 problem 文件中也要初始化声明某些 location 是 not_guarded 的。

还有一点就是，所有的解析最后得到的结果都是若干字符串组成的元组 tuple，如样例 1 中的初始状态谓词 (border town field) 就会被解析成 (”border”, ”town”, field)，objects 中的 npc - player 就会被解析成 (”npc”, ”player”)，具体使用这些结果时才会重新解释各自位置的意义。

在 heuristic 启发式函数部分，输入是一个由若干谓词组成的状态 state 列表，按照上一节提到的分层结构，就是循环中每次取出最后一个状态层，遍历所有可执行的 action，并将不包含在原先状态 state 中的 adds 谓词加进去即可，直到状态层包含了目标状态 goal，此时调用 CountAction 函数得到返回值返回即可。而 CountAction 函数部分，具体流程也就是上一节所提到的那样，每一次拆开目标状态 G 为 G_P 和 G_N，然后遍历所有可执行的 action，找出最小操作子集，取出它们的先决条件和 G_P 并在一起传给下一层递归即可。

在 Astar 算法部分就没有太多可以值得提及的点了，毕竟此前多次实验都有用到这个算法，创建一个新的类 class，定义这个类的__lt__内建函数，就可以使用 queue 库中的优先队列 PriorityQueue 来维护我们的状态队列，每次取出队列头部的一个状态，遍历所有可执行的 action，执行后加入队列即可。

4.1.3 Speedup Tricks

从上一节的描述中，其实我们很容易发现，在多个部分都要做同一个操作，那就是对某个状态枚举在该状态下可以执行的所有操作，我们不仅需要这个操作的序号名字，还需要可执行该操作的参数是什么，具体做法就是利用 objects 中定义的每个种类的物体，取它们的全排列依次检查是否满足先决条件。

基于这一点，我将这个操作独立出来并且建立了一个储存列表，相同状态 state 只要调用过一次这个函数，下一次再次调用时就直接返回结果，不需要再重新计算，这个计算实际上是相当耗时的，事实证明也是如此. 下面图 1 是我在第一版实现不使用该函数的执行效果（以样例 2 为测试基准，其他样例过于简单）。经过优化后执行效果如下图 2 所示，可以看到将近加速了一半，可以看出这确实是我的 implementation 中的一个性能瓶颈。

```
******************** Action List ********************
move ('npc', 'town', 'field')
attack ('npc', 'ogre', 'field', 'river')
move ('npc', 'field', 'river')
open ('npc', 'box1', 'river')
collect_fire ('npc', 'box1', 'river', 'reddust')
attack ('npc', 'dragon', 'river', 'cave')
move ('npc', 'river', 'cave')
open ('npc', 'box2', 'cave')
collect_earth ('npc', 'box2', 'cave', 'browndust')
build-fireball ('npc',)
************************ end ************************
Parser time cost:0.0069811344146728516
Astar search time cost:0.10470461845397949
```

Figure 1: 未优化的样例 2 执行效果

```
******************** Action List ********************
move ('npc', 'town', 'field')
attack ('npc', 'ogre', 'field', 'river')
move ('npc', 'field', 'river')
open ('npc', 'box1', 'river')
collect_fire ('npc', 'box1', 'river', 'reddust')
attack ('npc', 'dragon', 'river', 'cave')
move ('npc', 'river', 'cave')
open ('npc', 'box2', 'cave')
collect_earth ('npc', 'box2', 'cave', 'browndust')
build-fireball ('npc',)
************************ end ************************
Parser time cost:0.005018949508666992
Astar search time cost:0.04883933067321777
```

Figure 2: 优化后的样例 2 执行效果

4.1.4 TestCase

上一节图 2 已经展示了优化后的样例 2 执行效果，下面图 3-图 6 分别是优化后的样例 0, 样例 1, 样例 3 和样例 4 执行效果。从最终结果来看，其实除了样例 2 以外，其他四个样例的问题本身都比较简单，从 A 星算法搜索时间上已经看不出太大的区别，时间都限制在了 0.01s 以内。回到问题本身也可以很容易地发现，输出的 action 序列都是最佳方案，并不存在冗余的动作或其他错误动作。除此之外，五个测试样例中我都添加了一些为使用 STRIPS 语言而构建的谓词，具体修改的部分可以在报告所在文件夹查看附加的 pddl 文件。

```
******************** Action List ********************
move ('npc', 'town', 'field')
move ('npc', 'field', 'castle')
************************ end ************************
Parser time cost:0.0020248889923095703
Astar search time cost:0.0019648075103759766
```

Figure 3: 优化后的样例 0 执行效果

```
******************** Action List ********************
move ('npc', 'town', 'tunnel')
move ('npc', 'tunnel', 'river')
move ('npc', 'river', 'castle')
************************ end ************************
Parser time cost:0.001997232437133789
Astar search time cost:0.009995460510253906
```

Figure 4: 优化后的样例 1 执行效果

```
******************** Action List ********************
unstack ('b', 'a', 'x')
move ('b', 'x', 'y')
move ('a', 'x', 'y')
stack ('a', 'b', 'y')
************************ end ************************
Parser time cost:0.001996755599975586
Astar search time cost:0.003728151321411133
```

Figure 5: 优化后的样例 3 执行效果

```
******************** Action List ********************
unstack ('b', 'x', 'a', 'z')
stack ('a', 'x', 'b', 'z')
************************ end ************************
Parser time cost:0.0020325183868408203
Astar search time cost:0.00842905044555664
```

Figure 6: 优化后的样例 4 执行效果

4.2 Diagnosing by Bayesian Networks

4.2.1 VE algorithm

在贝叶斯网络这类概率图模型中，我们最关注的一个核心问题就是所谓的边缘推断，当我们总结出模型中一些变量的所有可能情况时，怎么知道其它的某些特定变量取某些特定值的概率是多少呢？

边缘推断问题将概率图中所有变量分为了三类，分别是已知变量 evidence variables，查询变量 query variables 和其它变量 remaining variables，变量消除法的思想十分简单，下面是它的大致工作流程：

1. 将所有的已知变量 evidence variables 直接代入每个节点的条件概率表 cpt 中，例如上次 VE 实验中的某个变量的 cpt 如下所示 ($P(J|A)$)：

   $$\{'11':0.9,'01':0.1,'10':0.05,'00':0.95\}$$

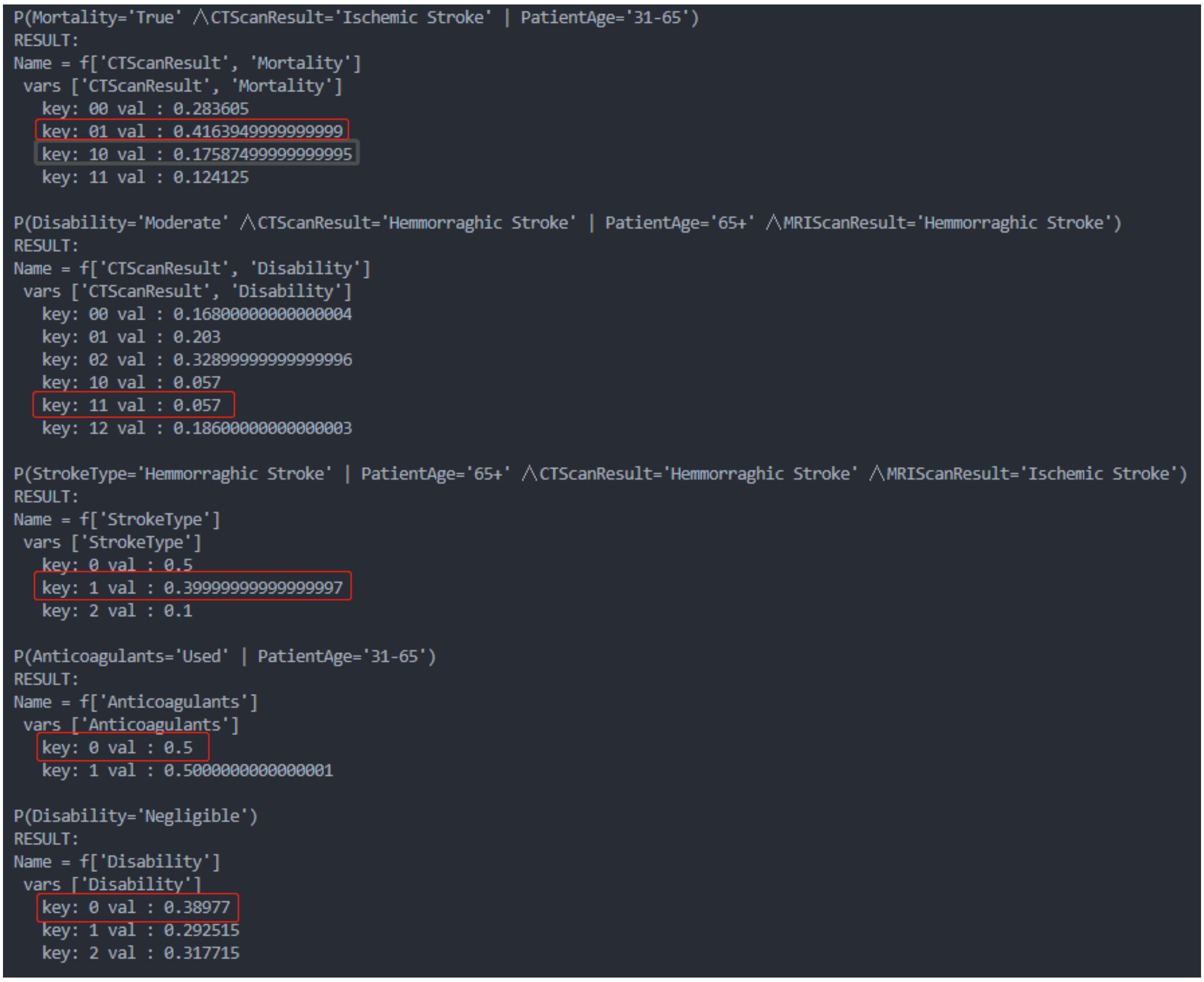
   如果已知变量 J 取 1，那么我们就可以将第一个数字（对应变量 J）是 1 的项挑选出来，也即第一个和第三个，那新的 cpt 就变成了

   $$\{'1':0.9,'0':0.05\}$$

   这个 cpt 里面的变量列表也就从 J,A 两个变量变为只剩下 A 一个变量了。
2. 按照一定的顺序依次消除所有的其它变量 remaining variables，首先初始化时我们根据代入已知变量后的所有条件概率可以获得若干因子，这些因子对应的变量列表中的变量要么是其它变量 remaining variables，要么是查询变量 query variables，消除某个变量就是将所有带该变量的因子取出来然后对该变量所有情况进行求和，再删去原先的这些因子并创建新的因子即可，新的因子变量列表就是原有的变量列表合并后去掉该被消除的变量。
3. 最后剩下的因子就是查询变量 remaining variables 的所有情况的概率，将这个作为最后结果返回。

4.2.2 Implementation

最基础的 VE algorithm 实现来自于实验 E09 的代码，因为当时已经对算法本身以及节点等进行封装，只需按照一些特定的函数调用输入我们的问题即可，注意此时的消除变量选取顺序是根据输入的变量列表来实现的，没有使用任何算法，下一节才会添加消除变量选取的一些算法来提升效率，下图 7 是五个概率值的计算结果，其中红框圈出来的是我们需要的 key 值。



```
P(Mortality='True' ∧CTScanResult='Ischemic Stroke' | PatientAge='31-65')
RESULT:
Name = f['CTScanResult', 'Mortality']
 vars ['CTScanResult', 'Mortality']
   key: 00 val : 0.283605
   key: 01 val : 0.4163949999999999
   key: 10 val : 0.17587499999999995
   key: 11 val : 0.124125

P(Disability='Moderate' ∧CTScanResult='Hemmorraghic Stroke' | PatientAge='65+' ∧MRIScanResult='Hemmorraghic Stroke')
RESULT:
Name = f['CTScanResult', 'Disability']
 vars ['CTScanResult', 'Disability']
   key: 00 val : 0.16800000000000004
   key: 01 val : 0.203
   key: 02 val : 0.32899999999999996
   key: 10 val : 0.057
   key: 11 val : 0.057
   key: 12 val : 0.18600000000000003

P(StrokeType='Hemmorraghic Stroke' | PatientAge='65+' ∧CTScanResult='Hemmorraghic Stroke' ∧MRIScanResult='Ischemic Stroke')
RESULT:
Name = f['StrokeType']
 vars ['StrokeType']
   key: 0 val : 0.5
   key: 1 val : 0.39999999999999997
   key: 2 val : 0.1

P(Anticoagulants='Used' | PatientAge='31-65')
RESULT:
Name = f['Anticoagulants']
 vars ['Anticoagulants']
   key: 0 val : 0.5
   key: 1 val : 0.5000000000000001

P(Disability='Negligible')
RESULT:
Name = f['Disability']
 vars ['Disability']
   key: 0 val : 0.38977
   key: 1 val : 0.292515
   key: 2 val : 0.317715
```

Figure 7: 未优化的 VE 算法运行结果

4.2.3 order selection algorithm

消除变量选取的算法有多种启发式规则，这里我采用的方法是，选取邻接边也就是相连节点最少的变量优先消除，实现算法的代码也比较简单，就是在仍未消除的所有变量中进行枚举，在 factor 因子中累计某个变量出现的次数即可，具体代码如下所示，最终运行结果也一并贴出如图 8，可以看出和原版的计算结果是相同的（小数点精度有损失）。

```
def select(orderedListOfHiddenVariables, factorList):
min_var = None
min_cnt = None
for var in orderedListOfHiddenVariables:
    cnt = 0
    for factor in factorList:
        if var in factor.varList:
```

```
            cnt += 1

    if min_cnt is None or cnt < min_cnt:
        min_var = var
        min_cnt = cnt

return min_var
```

```
P(Mortality='True' ∧CTScanResult='Ischemic Stroke' | PatientAge='31-65')
RESULT:
Name = f['CTScanResult', 'Mortality']
 vars ['CTScanResult', 'Mortality']
   key: 00 val : 0.28360500000000005
   key: 01 val : 0.416395
   key: 10 val : 0.175875
   key: 11 val : 0.124125

P(Disability='Moderate' ∧CTScanResult='Hemmorraghic Stroke' | PatientAge='65+' ∧MRIScanResult='Hemmorraghic Stroke')
RESULT:
Name = f['CTScanResult', 'Disability']
 vars ['CTScanResult', 'Disability']
   key: 00 val : 0.168
   key: 01 val : 0.203
   key: 02 val : 0.329
   key: 10 val : 0.05700000000000001
   key: 11 val : 0.05700000000000001
   key: 12 val : 0.18600000000000003

P(StrokeType='Hemmorraghic Stroke' | PatientAge='65+' ∧CTScanResult='Hemmorraghic Stroke' ∧MRIScanResult='Ischemic Stroke')
RESULT:
Name = f['StrokeType']
 vars ['StrokeType']
   key: 0 val : 0.5
   key: 1 val : 0.4
   key: 2 val : 0.1

P(Anticoagulants='Used' | PatientAge='31-65')
RESULT:
Name = f['Anticoagulants']
 vars ['Anticoagulants']
   key: 0 val : 0.5
   key: 1 val : 0.5

P(Disability='Negligible')
RESULT:
Name = f['Disability']
 vars ['Disability']
   key: 0 val : 0.38977
   key: 1 val : 0.292515
   key: 2 val : 0.317715
```

Figure 8: 优化后的 VE 算法运行结果

4.2.4 different orders of variable elimination

实验结果如下表所示，其中 p4 和 p5 的第一行都是优化算法得出的序列，因为五个变量的组合比较多，在尝试了若干组合后，最终选出了比较有代表性的几组用来展示，由于程序运行速度较快，展示的 total time 是一百组运算的时间和。

从结果数字上来看，一方面我所实现的选取算法无论是从时间上还是从 elimination width 上来看都可以并列在最优中，另一方面可以明显地发现运行时间和 elimination width 有着极大的关系，尤其是在样例 5 中可以看出来，当 width 在 3 的时候（也就是初始图的 width，说明在消除过程中因子的数量都没有超过初始值），运行时间只有 0.18s 左右，但是当 width 来到 6 时，运行时间几乎翻倍，width 为 4 时也比 3 的时候要高一些。

除此之外，我们可以发现当 width 相同的时候，其实变量顺序似乎没有那么重要，从样例 4 来看，相同的 width 在不同的顺序运行时间相差无几，所以我认为可以得出一个比较初步的结论就是，VE 算法的运行效率主要与 elimination width 相关，而在 width 相同的情况下，elimination order 没有什么影响。

Test case	Elimination order	Elimination width	Total time
p4	MO/DI/CT/MR/ST	3	0.15s
p4	CT/MR/ST/MO/DI	3	0.16s
p4	MR/CT/ST/MO/DI	3	0.16s
p4	MR/ST/CT/MO/DI	4	0.18s
p4	MR/ST/MO/CT/DI	4	0.18s
p4	MR/ST/MO/DI/CT	4	0.18s
p5	MO/PA/CT/MR/AN/ST	3	0.18s
p5	PA/CT/MR/AN/ST/MO	3	0.18s
p5	AN/PA/CT/MR/ST/MO	3	0.18s
p5	ST/AN/PA/CT/MR/MO	6	0.36s
p5	ST/AN/PA/MR/CT/MO	6	0.36s
p5	PA/MR/ST/AN/CT/MO	4	0.21s

5 Code

5.1 STRIPS planner

”strips_planner.py

```
# Author: Lu Yanzuo
# Date: 2020-11-22
# Description: a STRIPS planner by using A* search and heuristic
    function
#   implemented personally

```

```
from itertools import product

import re
import time
import queue


# parse parameters line
def parameters_parser(s):
    pattern = re.compile(r"\?(\w+) - (\w+)")
    return re.findall(pattern, s)[0]


# parse predicate/objects
def general_parser(s):
    pattern = re.compile(r"(\w+)")
    return tuple(re.findall(pattern, s))


# readin and parse domain file
def domain_parser(filename):
    name = []
    parameters = []
    prec = []
    adds = []
    dels = []

    with open(filename, "r") as domain_file:
        lines = domain_file.readlines()
        pos = 0

        while True:
```

```
if pos == len(lines):
    break

if "action" not in lines[pos]:
    pos += 1
    continue

line = lines[pos].strip() # action
name_index = line.find("action") + 7
name.append(line[name_index:])

pos += 1 # parameters
line = lines[pos].strip()
para_pattern = re.compile(r"(?:\?\w+ - \w+)")
para_list = re.findall(para_pattern, line)
temp_parameters = []
for para in para_list:
    temp_parameters.append(parameters_parser(para))
parameters.append(temp_parameters)

pos += 1 # precondition
line = lines[pos].strip()
prec_pattern = re.compile(r"\((?!not \()(?:\w+)(?: \?\w+)+\)
  (?!\))")
prec_list = re.findall(prec_pattern, line)
temp_prec = []
for prec_item in prec_list:
    temp_prec.append(general_parser(prec_item))
prec.append(temp_prec)

pos += 1 # effect
line = lines[pos].strip()
adds_pattern = re.compile(r"\((?!not \()(?:\w+)(?: \?\w+)+\)
```

```
                (?!\))")
            dels_pattern = re.compile(r"\((?:not )\((?:\w+)(?: \?\w+)+\)
                (?:\))")
            adds_list = re.findall(adds_pattern, line)
            dels_list = re.findall(dels_pattern, line)
            temp_adds = []
            temp_dels = []
            for _adds in adds_list:
                temp_adds.append(general_parser(_adds))
            for _dels in dels_list:
                temp_dels.append(general_parser(_dels)[1:])
            adds.append(temp_adds)
            dels.append(temp_dels)

            pos += 1

    return name, parameters, prec, adds, dels


# readin and parse problem file
def problem_parser(filename):
    objects = []
    init = []
    goal = []

    with open(filename, "r") as problem_file:
        lines = problem_file.readlines()
        pos = 0

        while True:
            if pos == len(lines):
                break
```

```
        line = lines[pos].strip()

        if "objects" in line:
            pos += 1
            line = lines[pos].strip()
            while line != ")":
                if line:
                    objects.append(general_parser(line))
                pos += 1
                line = lines[pos].strip()
        elif "init" in line:
            pos += 1
            line = lines[pos].strip()
            while line != ")":
                if line:
                    init.append(general_parser(line))
                pos += 1
                line = lines[pos].strip()
        elif "goal" in line:
            goal_pattern = re.compile(r"\((?!not \()(?:\w+)(?: \w+)
               +\)(?!\))")
            goal_list = re.findall(goal_pattern, line)
            for goal_item in goal_list:
                goal.append(general_parser(goal_item))

        pos += 1

    return objects, init, goal


# replace predicate variable with real paras
def replace(prec, para_variable, paras):
    # start_time = time.time()
```

```
    result = [prec[0]]
    for prec_item in prec[1:]:
        index = para_variable.index(prec_item)
        result.append(paras[index])
    # end_time = time.time()
    return tuple([*result])



# store_input = []
# store_output = []
# check whether action is valid for current state
def check_action(cur_state, parameters, prec, objects):
    # start_time = time.time()
    # # print("check action cur_state:", cur_state)

    # if (cur_state, parameters, prec) in store_input:
    #     return store_output[store_input.index((cur_state, parameters,
        prec))]

    objects_name = []
    objects_item = []
    for _object in objects:
        objects_name.append(_object[-1])
        objects_item.append([*_object[:-1]])

    para_index = []
    para_variable = []
    for para in parameters:
        para_index.append(objects_name.index(para[1]))
        para_variable.append(para[0])

    para_item = []
    for i in para_index:
```

```
            para_item.append(objects_item[i])

    result = []
    for paras in product(*para_item):
        if len(set(paras)) < len(paras):
            continue
        valid_flag = True
        for prec_item in prec:
            replace_pred = replace(prec_item, para_variable, paras)
            if replace_pred not in cur_state:
                valid_flag = False
                break
        if valid_flag:
            result.append(paras)

    # store_input.append((cur_state, parameters, prec))
    # store_output.append(None if len(result) == 0 else result)

    # end_time = time.time()
    # # print("Check Action function takes time {}s".format(end_time -
        start_time))
    return None if len(result) == 0 else result



# get valid action index and paras for specific state
store_input = []
store_output = []
def get_valid_action(state):
    global name
    global parameters
    global prec
    global objects
```

```python
    if state in store_input:
        return store_output[store_input.index(state)]

    valid_action_index = []
    valid_action_paras = []
    for action_index in range(len(name)):
        res = check_action(state, parameters[action_index], \
            prec[action_index], objects)
        if res is not None:
            valid_action_index.append(action_index)
            valid_action_paras.append(res)

    store_input.append(state)
    store_output.append((valid_action_index, valid_action_paras))
    return valid_action_index, valid_action_paras


# CountAction recursive function
def CountAction(goal, state_layer, cur_layer, action, objects):
    # start_time = time.time()
    # print("cur_layer:", cur_layer)

    if cur_layer == 0:
        return 0

    # print("cur state layer:", state_layer[cur_layer])
    # print("pre state layer:", state_layer[cur_layer-1])

    name = action["name"]
    parameters = action["parameters"]
    prec = action["prec"]
    adds = action["adds"]
```

```
goal_p = []
goal_n = []

for goal_item in goal:
    if goal_item in state_layer[cur_layer-1]:
        goal_p.append(goal_item)
    else:
        goal_n.append(goal_item)

# print("goal_p:", goal_p)
# print("goal_n:", goal_n)

objects_name = []
objects_item = []
for _object in objects:
    objects_name.append(_object[-1])
    objects_item.append([*_object[:-1]])

# valid_action_index = []
# valid_action_paras = []
# for action_index in range(len(name)):
#     res = check_action(state_layer[cur_layer-1], parameters[
    action_index], \
#         prec[action_index], objects)
#     if res is not None:
#         valid_action_index.append(action_index)
#         valid_action_paras.append(res)

valid_action_index, valid_action_paras = get_valid_action(
    state_layer[cur_layer-1])

# print("valid_action_index:", valid_action_index)
# print("valid_action_paras:", valid_action_paras)
```

```python
action_index_set = []
action_paras_set = []
adds_list = []
for i in range(len(valid_action_index)):
    para_variable = []
    for para in parameters[valid_action_index[i]]:
        para_variable.append(para[0])

    for paras in valid_action_paras[i]:
        temp_adds_list = []
        for pred in adds[valid_action_index[i]]:
            replace_pred = replace(pred, para_variable, paras)
            temp_adds_list.append(replace_pred)

        for temp_adds in temp_adds_list:
            if temp_adds not in adds_list and temp_adds in goal_n:
                action_index_set.append(valid_action_index[i])
                action_paras_set.append(paras)
                adds_list.extend(temp_adds_list)
                break

        if set(goal_n) <= set(adds_list):
            break

# print("action_index_set:", action_index_set)
# print("action_paras_set:", action_paras_set)
# # print("adds_list:", adds_list)

next_goal = goal_p.copy()
for i in range(len(action_index_set)):
    para_index = []
    para_variable = []
```

```
        for para in parameters[action_index_set[i]]:
            para_index.append(objects_name.index(para[1]))
            para_variable.append(para[0])

        for prec_item in prec[action_index_set[i]]:
            replace_pred = replace(prec_item, para_variable,
                action_paras_set[i])
            if replace_pred not in next_goal:
                next_goal.append(replace_pred)

    # print("next_goal:", next_goal)

    # end_time = time.time()
    # # print("Count Action function takes time {}s".format(end_time -
        start_time))
    return CountAction(next_goal, state_layer, cur_layer-1, action,
        objects) + len(action_index_set)


# compute heuristic function values for current state
def heuristic(cur_state, action, goal, objects):
    start_time = time.time()

    state_layer = []
    state_layer.append(cur_state)

    name = action["name"]
    parameters = action["parameters"]
    prec = action["prec"]
    adds = action["adds"]

    while True:
        # print("the latest state layer:", state_layer[-1])
```

```
goal_flag = True
for goal_item in goal:
    if goal_item not in state_layer[-1]:
        goal_flag = False
        break
if goal_flag:
    # print("get goal state layer.")
    break

# valid_action_index = []
# valid_action_paras = []
# for action_index in range(len(name)):
#     res = check_action(state_layer[-1], parameters[
    action_index], \
#         prec[action_index], objects)
#     if res is not None:
#         valid_action_index.append(action_index)
#         valid_action_paras.append(res)

valid_action_index, valid_action_paras = get_valid_action(
    state_layer[-1])

# print("valid action index and paras:", valid_action_index,
    valid_action_paras)
# print("\n")

next_state = state_layer[-1].copy()

for i in range(len(valid_action_index)):
    para_variable = []
    for para in parameters[valid_action_index[i]]:
        para_variable.append(para[0])
```

```
            for valid_paras in valid_action_paras[i]:
                for pred in adds[valid_action_index[i]]:
                    replace_pred = replace(pred, para_variable,
                        valid_paras)
                    if replace_pred not in next_state:
                        next_state.append(replace_pred)

        state_layer.append(next_state)

    heuristic_value = CountAction(goal, state_layer, len(state_layer)-1,
        action, objects)

    end_time = time.time()
    # print("Heuristic function takes time {}s".format(end_time -
        start_time))
    return heuristic_value


class Node(object):
    def __init__(self, state, gx, action_index, action_paras):
        self.state = state
        self.gx = gx
        self.hx = heuristic(state, action, goal, objects)
        self.fx = self.gx + self.hx
        self.action_index = action_index.copy()
        self.action_paras = action_paras.copy()

    def __lt__(self, other):
        return self.fx < other.fx

    def is_goal(self):
        return set(goal) <= set(self.state)
```

```
if __name__ == "__main__":
    test_num = int(input("Please input the number of test file(0-4): "))

    parser_start = time.time()

    domain_filename = "pddl/test{}/test{}_domain.txt".format(str(
        test_num), str(test_num))
    problem_filename = "pddl/test{}/test{}_problem.txt".format(str(
        test_num), str(test_num))

    name, parameters, prec, adds, dels = domain_parser(domain_filename)
    objects, init, goal = problem_parser(problem_filename)
    action = {
        "name": name,
        "parameters": parameters,
        "prec": prec,
        "adds": adds,
        "dels": dels
    }

    history = []
    q = queue.PriorityQueue()

    history.append(init)
    init_node = Node(init, 0, [], [])
    q.put(init_node)

    parser_end = time.time()

    while not q.empty():
        head_node = q.get()
```

```
if head_node.is_goal():
    print("\n" + "*****"*4 + " Action List " + "*****"*4)
    for i in range(len(head_node.action_index)):
        print(action["name"][head_node.action_index[i]],
            head_node.action_paras[i])
    print("*****"*4 + "**** end ****" + "*****"*4)
    break

# print("head_node.state:", head_node.state)

# valid_action_index = []
# valid_action_paras = []
# for action_index in range(len(name)):
#     res = check_action(head_node.state, parameters[
    action_index], \
#         prec[action_index], objects)
#     if res is not None:
#         valid_action_index.append(action_index)
#         valid_action_paras.append(res)

valid_action_index, valid_action_paras = get_valid_action(
    head_node.state)

for i in range(len(valid_action_index)):
    para_variable = []
    for para in parameters[valid_action_index[i]]:
        para_variable.append(para[0])

    for valid_paras in valid_action_paras[i]:
        new_state = head_node.state.copy()

        for pred in dels[valid_action_index[i]]:
            replace_pred = replace(pred, para_variable,
```

```
                valid_paras)
            if replace_pred in new_state:
                new_state.remove(replace_pred)

        for pred in adds[valid_action_index[i]]:
            replace_pred = replace(pred, para_variable,
                valid_paras)
            if replace_pred not in new_state:
                new_state.append(replace_pred)

        if new_state not in history:
            history.append(new_state)
            # print("new_state:", new_state)

            new_index = head_node.action_index.copy()
            new_paras = head_node.action_paras.copy()

            new_index.append(valid_action_index[i])
            new_paras.append(valid_paras)
            # print("new_index:", new_index)
            # print("new_paras:", new_paras)

            new_node = Node(new_state, head_node.gx+1, new_index
                , new_paras)
            q.put(new_node)

astar_end = time.time()
print("Parser time cost:{}".format(parser_end - parser_start))
print("Astar search time cost:{}\n".format(astar_end - parser_end))
```

5.2 Diagnosing by Bayesian Networks

"ve_algorithm.py"

```python
from math import pow
import time

maximum_width = None

def select(orderedListOfHiddenVariables, factorList):
    min_var = None
    min_cnt = None
    for var in orderedListOfHiddenVariables:
        cnt = 0
        for factor in factorList:
            if var in factor.varList:
                cnt += 1

        if min_cnt is None or cnt < min_cnt:
            min_var = var
            min_cnt = cnt

    return min_var

class VariableElimination:
    @staticmethod
    def inference(factorList, queryVariables,
       orderedListOfHiddenVariables, evidenceList):
        global maximum_width

        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])
```

```python
    # for var in orderedListOfHiddenVariables:
    while len(orderedListOfHiddenVariables) > 0:
        var = select(orderedListOfHiddenVariables, factorList)
        orderedListOfHiddenVariables.remove(var)
        # print(var)
        #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)

        if len(new_factor.varList) > maximum_width:
            maximum_width = len(new_factor.varList)

        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]

        new_node = Node("f" + str(new_var_list), new_var_list)
        new_node.setCpt(new_cpt)
        return new_node

# create nodes for Bayes Net
PatientAge = Node("PatientAge", ["PatientAge"])
CTScanResult = Node("CTScanResult", ["CTScanResult"])
MRIScanResult = Node("MRIScanResult", ["MRIScanResult"])
Anticoagulants = Node("Anticoagulants", ["Anticoagulants"])
StrokeType = Node("StrokeType", ["StrokeType", "CTScanResult", "
   MRIScanResult"])
Mortality = Node("Mortality", ["Mortality", "StrokeType", "
   Anticoagulants"])
Disability = Node("Disability", ["Disability", "StrokeType", "PatientAge
   "])
```

```
# Generate cpt for each node
PatientAge.setCpt({'0': 0.1, '1': 0.3, '2': 0.6})
CTScanResult.setCpt({'0': 0.7, '1': 0.3})
MRIScanResult.setCpt({'0': 0.7, '1': 0.3})
Anticoagulants.setCpt({'0': 0.5, '1': 0.5})
StrokeType.setCpt({'000': 0.8, '001': 0.5, '010': 0.5, '011': 0.0,
    '100': 0.0, '101': 0.4, '110': 0.4, '111': 0.9,
    '200': 0.2, '201': 0.1, '210': 0.1, '211': 0.1})
Mortality.setCpt({'000': 0.28, '010': 0.99, '020': 0.1,
    '001': 0.56, '011': 0.58, '021': 0.05,
    '100': 0.72, '110': 0.01, '120': 0.9,
    '101': 0.44, '111': 0.42, '121': 0.95})
Disability.setCpt({'000': 0.8, '010': 0.7, '020': 0.9,
    '001': 0.6, '011': 0.5, '021': 0.4,
    '002': 0.3, '012': 0.2, '022': 0.1,
    '100': 0.1, '110': 0.2, '120': 0.05,
    '101': 0.3, '111': 0.4, '121': 0.3,
    '102': 0.4, '112': 0.2, '122': 0.1,
    '200': 0.1, '210': 0.1, '220': 0.05,
    '201': 0.1, '211': 0.1, '221': 0.3,
    '202': 0.3, '212': 0.6, '222': 0.8,})

print("P(Mortality='True'  CTScanResult='Ischemic Stroke' | PatientAge
   ='31-65')")
VariableElimination.inference([PatientAge, CTScanResult, MRIScanResult,
   Anticoagulants, \
    StrokeType, Mortality, Disability], ["Mortality", "CTScanResult"], \
    ["MRIScanResult", "Anticoagulants", "StrokeType", "Disability"], {"
       PatientAge": 1})

print("P(Disability='Moderate'  CTScanResult='Hemmorraghic Stroke' |
   PatientAge='65+'  MRIScanResult='Hemmorraghic Stroke')")
```

```
VariableElimination.inference([PatientAge, CTScanResult, MRIScanResult, Anticoagulants, \
    StrokeType, Mortality, Disability], ["Disability", "CTScanResult"], \
    ["Anticoagulants", "StrokeType", "Mortality"], {"PatientAge": 2, "MRIScanResult": 1})

print("P(StrokeType='Hemmorraghic Stroke' | PatientAge='65+' □ CTScanResult='Hemmorraghic Stroke' □ MRIScanResult='Ischemic Stroke')")
VariableElimination.inference([PatientAge, CTScanResult, MRIScanResult, Anticoagulants, \
    StrokeType, Mortality, Disability], ["StrokeType"], \
    ["Anticoagulants", "Mortality", "Disability"], \
    {"PatientAge": 2, "CTScanResult": 1, "MRIScanResult": 0})

print("P(Anticoagulants='Used' | PatientAge='31-65')")
VariableElimination.inference([PatientAge, CTScanResult, MRIScanResult, Anticoagulants, \
    StrokeType, Mortality, Disability], ["Anticoagulants"], \
    ["CTScanResult", "MRIScanResult", "StrokeType", "Mortality", "Disability"], \
    {"PatientAge": 1})

print("P(Disability='Negligible')")
VariableElimination.inference([PatientAge, CTScanResult, MRIScanResult, Anticoagulants, \
    StrokeType, Mortality, Disability], ["Disability"], \
    ["PatientAge", "CTScanResult", "MRIScanResult", "Anticoagulants", "StrokeType", "Mortality"], \
    {})

# start = time.time()
```

```
# for i in range(100):
#       print("P(Anticoagulants='Used' | PatientAge='31-65')")
#       maximum_width = 3
#       VariableElimination.inference([PatientAge, CTScanResult,
    MRIScanResult, Anticoagulants, \
#           StrokeType, Mortality, Disability], ["Disability"], \
#           ["PatientAge", "MRIScanResult","StrokeType","Anticoagulants",
      "CTScanResult", "Mortality"], \
#           {})
#       print(maximum_width)
# end = time.time()

# print("total time: {}".format(end - start))
```