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

P03 Planning and Uncertainty

18340161 田蕊 18340149 孙新梦

November 27, 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) **解答：**

   这个算法的核心思想就是在只考虑添加状态不考虑删除状态的松弛问题中考虑达到目标状态所需要动作数。**搜索问题**首先，在 STRIPS 规划中，我们把 Planning 看作一种搜索问题，状态表示为封闭世界知识库，初始的知识库是初始状态，动作则认为是把一个状态映射到另一个状态上，由于封闭世界假设，我们很容易验证目标是否在目前的知识库中，也就是是否已经达成。**松弛问题**但是问题在于当搜索树过大的时候会需要利用松弛问题的启发值，用启发式算法来确定下一个搜索的节点，才会比较快速找到我们的目标解，接下来我们阐述用可达性分析来获得启发式函数值的思想：

   当每个动作的前提都是正向的事实 (不是 Not), 并且目标也是正向事实，把原问题放松为动作的效果，只考虑 add 列表，而不考虑 delete 列表对问题的知识库的影响，则这样动作的前提更容易被满足，因此很多之前问题不可做的问题也可以做了。有定理指出：松弛问题的解的长度小于等于原问题的解的长度，因此是可采纳的。**可达性分析**计算启发式值的时候，我们用逐步建立状层态 S 和动作层 A 的方法，来获得我们在此松弛问题下，需要多少步动作能够到达目标，作为启发式值。

   首先开始于初始状态 S_0，接下来找所有前提已经在 S_0 中的动作，组成 A_0，下一层的 S_1 是 S_0 并上 A_0 的 add 效果列表，之后继续这个过程。其中 A_i 表示前提在 S_i 中，但是动作没有出现在上一个动作层 A_{i-1} 的动作，而状态层 S_i 是上一个动作层和本次能新进行的动作添加列表的并集。

   最后直到目标全部含于当前状态层，或者状态层不再改变，则终止算法。如果这个时候状态层没有目标，则不可达；反之，就可达，并且此时的动作数就是启发式函数值。

   动作数的计算方法如下：首先找出能够涵盖目标状态的最小动作集 A，把当前状态层划分为该动作集的添加效果 G_N 其他部分为第二部分,G_P, 后者和 A 的前提集合的并集作为下一回的调用参数，接下来继续在 size(A) 的基础上加上递归调用此函数在上一层的动作数。

2. Implement a STRIPS planner by using A* search and the heuristic function you implemented.(20

points) **思路**我们为了代码的整洁性，在数据结构方面，用一个大类 Problem 来涵盖整个搜索问题的相关函数及变量，包括当前的状态，目标状态，以及动作列表等。并把搜索函数，寻找可用的动作等函数定义在里面。Action 类作为一个内部类定义在 Problem 中，含有前提，添加，删除等列表及其他变量，以及判断此动作当前是否可做，更新状态，回溯状态等操作。

这里我们的逻辑是这样的，首先由 main 调用函数，获取初始状态可做的动作，然后进入 solve 函数，寻找可做的动作，并用启发式值和 cost 排序为优先级队列，取队首动作执行，并更新状态，递归继续搜索过程，若判断可做动作为空，则回退一个动作。

递归搜索函数

```
    def solve(self, actions):
        # 获取第一个动作
        theAction = actions.get()[1]
        aaaaaction = [theAction.name]
        for val in theAction.instance.values():
            aaaaaction.append(val)
        # print(aaaaaction)
        # 做这个动作
        theAction.do()
        self.cost += 1
        Problem.solves.append(theAction)
        is_goal = True
        for goal in Problem.goalState:
            if goal not in Problem.nowState:
                is_goal = False
        if is_goal:
            return True
        # if Problem.goalState in Problem.nowState:
        #     return True

        actionsb = self.changeActionstoQueue()
        if actionsb:
            return self.solve(actionsb)
        # 如果运行到这了就说明这个动作会导致问题无解，回溯
        theAction.pullBack()
```

```
26          self.cost -= 1
27          Problem.solves.remove(theAction)
```

3. Explain any ideas you use to speed up the implementation. (10 points)

   (a) 在判断动作是否可做的时候，我们没有去遍历动作含有的变量的类型，再遍历该类型所有实例化，来构造实例化新的 Action 对象，再去判断前提是否涵盖于当前状态。

   而是选择去遍历所有 Problem 文件出现的所有实例化的变量，不管类型，做一个排列，之后对每个排列来判断类型和动作变量表是否相匹配。原因在于我们的问题中，实例化的变量个数还算较少，遍历他们的排列比起遍历类型来说更迅速。

   如 [move,p,l1,l2],(p-player,l1-location,l2-location), 读文件之后，player 只有 npc 一个，location 有 town,field,castle 三个，不去根据 player、location 去找对应的 npc,town,field,castle，而是通过如 [npc,town,field] 这样的排列判断是否符合类型。

   (b) 在于对前提是否在状态的判断，我们并非全部实例化完动作的 pre,add,del 之后再判断，而是对前提的每一项，挨个实例化，判断是否存在，一旦有一个不存在，就判断出不可做。不再继续，break 出来，减少不必要的实例化。

   (c) 在寻找启发式函数值最小的动作的时候，使用优先队列而不是遍历查找。优先队列默认采用小根堆的数据结构，插入的时间复杂度是 O(logN), 而获取最小值的时间复杂度是 O(1) 的，这相对于 O(N) 的时间复杂度是一个很大的优化。

   (d) 整个代码使用了软件设计模式中的单例模式，在这个模式下，单例对象的类 Problem 只有一个实例存在。这样有利于我们协调系统的整体的行为。在单例模式中，活动的单例只有一个实例，对单例类的所有实例化得到的都是相同的一个实例。这样就防止其它对象对自己的实例化，确保所有的对象都访问一个实例。由于在系统内存中只存在一个对象，因此可以节约系统资源，当需要频繁创建和销毁的对象时单例模式无疑提高了系统的性能。同时这还可以避免对共享资源的多重占用。

4. Run you planner on the 5 test cases, and report the returned plans and the running times. Analyse the experimental results. (10 points) **test case 0**

```
-------Actions-------
action 0  : ['move', 'npc', 'town', 'field']
action 1  : ['move', 'npc', 'field', 'castle']
--------Done--------

Process finished with exit code 0
```

Figure 1: test0 result

```
-------Actions-------
action 0  : ['move', 'npc', 'town', 'tunnel']
action 1  : ['move', 'npc', 'tunnel', 'river']
action 2  : ['move', 'npc', 'river', 'castle']
--------Done--------

Process finished with exit code 0
```

Figure 2: test2 result

```
-------Actions-------
action 0  : ['move', 'npc', 'town', 'field']
action 1  : ['attack', 'npc', 'ogre', 'field', 'river']
action 2  : ['move', 'npc', 'field', 'river']
action 3  : ['open', 'npc', 'box1', 'river']
action 4  : ['collect-fire', 'npc', 'box1', 'river', 'reddust']
action 5  : ['attack', 'npc', 'dragon', 'river', 'cave']
action 6  : ['move', 'npc', 'river', 'cave']
action 7  : ['open', 'npc', 'box2', 'cave']
action 8  : ['collect-earth', 'npc', 'box2', 'cave', 'browndust']
action 9  : ['build-fireball', 'npc']
--------Done--------

Process finished with exit code 0
```

Figure 3: test2 result

```
-------Actions-------
action 0  : ['unstack', 'b', 'a', 'x']
action 1  : ['move', 'b', 'x', 'y']
action 2  : ['move', 'a', 'x', 'y']
action 3  : ['stack', 'a', 'b', 'y']
--------Done--------

Process finished with exit code 0
```

Figure 4: test3 result

```
-------Actions-------
action 0  : ['unstack', 'b', 'a', 't1', 't3']
action 1  : ['stack', 'a', 't1', 'b', 't3']
--------Done--------

Process finished with exit code 0
```

Figure 5: test4 result

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

Note: [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)

VE algorithm 在计算给定证据下某些变量的结果的时候，我们要应用链式法则和全概公式来计算，其中链式法则将表达式拆解成乘积的形式，而全概公式将表达式拆解成求和的形式。再通过限制住某些元素得到取值来得到我们最终想要的结果。VE 算法通过 restrict(限制变量的取值)，multiply(进行链式法则实现概率之间的相乘)，sumout(全概公式求和) 三个操作，将无关变量通过相乘求和等操作消去，留下在证据下我们需要查询的变量的概率。

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')

实现思路 有了实验八的基础，这一部分的代码只需要在实验八的基础上稍作修改即可。首先我们添加了一个名为 valueMap 的字典，里面讲每一个变量的可取值映射为 0，1，2……这样做可以方便后续的数值选取。其次我们将原本传入的列表类型 queryVariables 换成了字典类型变量，并在其中记录每一个查询变量的查询取值，这样在输出结果的时候，只需要输出查询的取值结果而不需要输出全部的结果，相应地在 printInf 函数中做出一些格式化的调整。下面我们分别给出我们的程序计算的概率结果数值和使用 pomegranate 进行计算得到的结果，经过比对可以发现二者在误差可允许的范围内结果一致。会出现微小的误差是由变量消除的顺序导致的，因为消除的顺序不同所以计算的结果运算的顺序也不同。

```
our code's result:
RESULT:
0.41639499999999996
max eliminate width: 6
Elimination order :  ['StrokeType', 'Disability', 'MRIScanResult', 'Anticoagulants']

RESULT:
0.057
max eliminate width: 6
Elimination order :  ['StrokeType', 'Anticoagulants', 'Mortality']

RESULT:
0.4
max eliminate width: 3
Elimination order :  ['Disability', 'Mortality', 'Anticoagulants']

RESULT:
0.5
max eliminate width: 6
Elimination order :  ['StrokeType', 'Disability', 'Mortality', 'CTScanResult', 'MRIScanResult']

RESULT:
0.38977
max eliminate width: 6
Elimination order :  ['StrokeType', 'Mortality', 'PatientAge', 'Anticoagulants', 'CTScanResult', 'MRIScanResult']
```

Figure 6: our code's result

```
pomegranate result:
p1 =  0.41639499999999974
p2 =  0.05700000000000005
p3 =  0.3999999999999996
p4 =  0.5
p5 =  0.38977
```

Figure 7: pomegranate result

3. Implement an algorithm to select a good order of variable elimination. (10 points)

PolyTree 根据给出的概率表，我们可以知道我们的这个问题是一个多叉树，变量之间的相互作用关系如下所示。对于贝叶斯网络中的多叉树来说，在每一步我们都消掉一个单一连接的节点 (因为是多叉树，所以一定能保证在每一步消去的时候都有单一连接的节点)，那么树的因子的大小将一直不会增加，也就是说，初始时因子中变量数目最大的一个即为树的宽度。

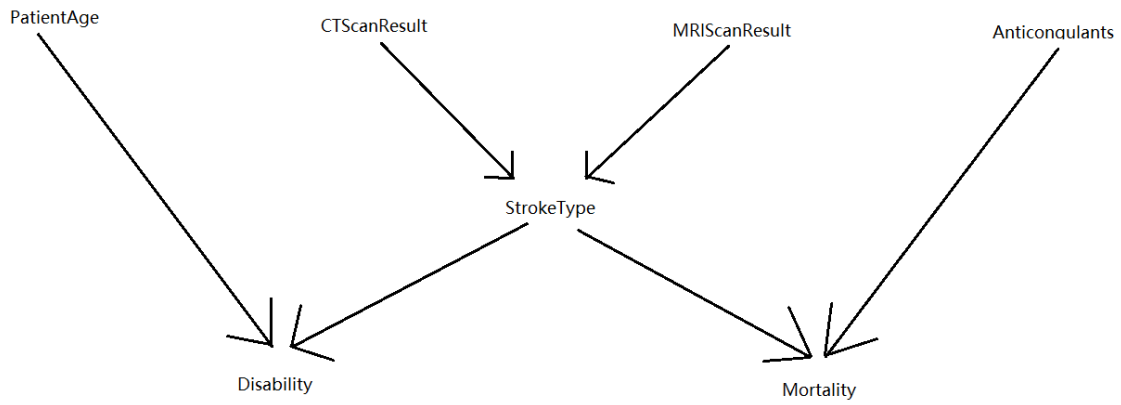

Figure 8: the polytree of the problem

Min-Fill Heuristic 我们使用的启发式策略是每次都消除产生最小因子的变量。这一算法在多树问题上是线性时间复杂度的。在多数问题中我们只需要在待消除变量中寻找单一连接的节点，然后在单一连接的节点中因子大小最小的变量先进行消除即可。这体现在我们的数据结构中只需要根据变量表的大小就可以判断。我们利用一个优先队列来帮助简化编程，具体代码如下所示。

```
def orderedVariables(factorList, orderedListOfHiddenVariables: list):
    q = queue.PriorityQueue()
    for v in orderedListOfHiddenVariables:
        for f in factorList:
            if v == f.name:
                q.put((len(f.varList), v))
                break
    orderedListOfHiddenVariables.clear()
    while q.qsize():
        orderedListOfHiddenVariables.append(q.get()[1])
```

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)

由于问题模型很简单，即使是使用最差的消除顺序运行时间也很短，为了使结果更加明显，我们将每个消除顺序都运行 5000 次后比对运行时间的差异。在测试时，注释掉程序的输出部分，将待消除变量保存到一个列表中，之后调用我们实现的 orderedVariables 函数寻找最优消除顺序，之后调用 random.shuffle(eliminatelist) 函数来对消除顺序进行随机重排后进行消除宽度和时间上的测试。

运行方法 为了方便检查和测试，我们的程序可以根据输入执行不同的功能，具体内容如下：

(a) 输入“1” 计算所有运算结果

(b) 输入“2” 比对不进行消除顺序判断和进行消除顺序判断两种情况下的运行时间差异（5000 次运行下的时间）

(c) 输入“3” 查看 P4 在不同计算顺序 (第一个顺序为算法找到的消除顺序) 下的运行时间（5000 次运行下的时间）

(d) 输入“4” 查看 P5 在不同计算顺序 (第一个顺序为算法找到的消除顺序) 下的运行时间（5000 次运行下的时间）

TA 在检查代码时只需要输入指令即可。

Test case	Elimination order	Elimination width	Total time
p4	3	['CTScanResult', 'MRIScanResult', 'Disability', 'Mortality', 'StrokeType']	1.23 s
p4	4	['MRIScanResult', 'Mortality', 'StrokeType', 'Disability', 'CTScanResult']	2.51 s
p4	4	['CTScanResult', 'MRIScanResult', 'StrokeType', 'Disability', 'Mortality']	2.84 s
p4	6	['StrokeType', 'Mortality', 'MRIScanResult', 'CTScanResult', 'Disability']	7.70 s
p4	5	['CTScanResult', 'StrokeType', 'MRIScanResult', 'Disability', 'Mortality']	4.74 s
p4	6	['StrokeType', 'Mortality', 'Disability', 'CTScanResult', 'MRIScanResult']	7.53 s
p5	3	['Anticoagulants', 'CTScanResult', 'MRIScanResult', 'PatientAge', 'Mortality', 'StrokeType']	2.10 s
p5	4	['MRIScanResult', 'PatientAge', 'StrokeType', 'Anticoagulants', 'Mortality', 'CTScanResult']	4.94 s
p5	5	['CTScanResult', 'StrokeType', 'PatientAge', 'Anticoagulants', 'Mortality', 'MRIScanResult']	12.89 s
p5	6	['StrokeType', 'Mortality', 'MRIScanResult', 'Anticoagulants', 'PatientAge', 'CTScanResult']	20.77s
p5	4	['Anticoagulants', 'PatientAge', 'StrokeType', 'MRIScanResult', 'Mortality', 'CTScanResult']	4.40 s
p5	6	['StrokeType', 'Anticoagulants', 'PatientAge', 'Mortality', 'CTScanResult', 'MRIScanResult']	22.61 s

结果截图展示

```
count p4 = P(Anticoagulants='Used' | PatientAge='31-65')

our order of variable elimination:  ['CTScanResult', 'MRIScanResult', 'Disability', 'Mortality', 'StrokeType']
running time =  1.2300028800964355  s
Elimination width =  3
random order of variable elimination:  ['MRIScanResult', 'Mortality', 'StrokeType', 'Disability', 'CTScanResult']
running time =  2.509963274002075  s
Elimination width =  4
random order of variable elimination:  ['CTScanResult', 'MRIScanResult', 'StrokeType', 'Disability', 'Mortality']
running time =  2.8402833938598633  s
Elimination width =  4
random order of variable elimination:  ['StrokeType', 'Mortality', 'MRIScanResult', 'CTScanResult', 'Disability']
running time =  7.700991153717041  s
Elimination width =  6
random order of variable elimination:  ['CTScanResult', 'StrokeType', 'MRIScanResult', 'Disability', 'Mortality']
running time =  4.739139795303345  s
Elimination width =  5
random order of variable elimination:  ['StrokeType', 'Mortality', 'Disability', 'CTScanResult', 'MRIScanResult']
running time =  7.532074213027954  s
Elimination width =  6
```

Figure 9: p4 result

```
count p5 = P(Disability='Negligible')

our order of variable elimination:  ['Anticoagulants', 'CTScanResult', 'MRIScanResult', 'PatientAge', 'Mortality', 'StrokeType']
running time =  2.107309579849243  s
Elimination width =  3
random order of variable elimination:  ['MRIScanResult', 'PatientAge', 'StrokeType', 'Anticoagulants', 'Mortality', 'CTScanResult']
running time =  4.941882610321045  s
Elimination width =  4
random order of variable elimination:  ['CTScanResult', 'StrokeType', 'PatientAge', 'Anticoagulants', 'Mortality', 'MRIScanResult']
running time =  12.89268684387207  s
Elimination width =  5
random order of variable elimination:  ['StrokeType', 'Mortality', 'MRIScanResult', 'Anticoagulants', 'PatientAge', 'CTScanResult']
running time =  20.77104425430298  s
Elimination width =  6
random order of variable elimination:  ['Anticoagulants', 'PatientAge', 'StrokeType', 'MRIScanResult', 'Mortality', 'CTScanResult']
running time =  4.402190208435059  s
Elimination width =  4
random order of variable elimination:  ['StrokeType', 'Anticoagulants', 'PatientAge', 'Mortality', 'CTScanResult', 'MRIScanResult']
running time =  22.612092971801758  s
Elimination width =  6
```

Figure 10: p5 result

结果分析 可以看出不同的变量消除顺序对于结果的影响是巨大的，而我们的算法找出的消除顺序无论是在消除宽度上还是运行时间上都有很好的效果。

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