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

P03 Planning and Uncertainty

18340013 陈琮昊 18340014 陈嘉宁

Due: 11:59pm, Saturday, Nov. 28, 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)
   给定一些 action 和 state 的相关信息, 我们现在想找到一系列动作能够到达目标, 可以采用启发式搜索的方法. 通过课上学习的内容可以知道, 我们可以解决原问题的松弛问题 (Relaxed Problem). 所谓的松弛问题就是不考虑 `STRIPS` 里 `delete` 运算符下的行为. 为什么有这样的结论? 因为我们证明了: 松弛问题的最优解长度以原问题的最优解长度为上界. 因此松弛问题的解长度可以作为一个 A* 的启发式函数. 但是如何求解一个松弛问题呢? 就用到了 `Reachability analysis`. 简单来说就是构建 `state layers` 和 `action layers`, 根据其性质进行运算, 直到某一层包含了目标状态. 这时我们再递归调用 `CountActions` 这个函数来获取解的长度从而作为启发式函数.

2. Implement a STRIPS planner by using A* search and the heuristic function you implemented.(20 points)
   这里只放出 A* 搜索的相关部分, 完整的 `STRIPS planner` 详见 `Appendix` 一栏里的 `planner.py`.

```
# 当前状态与目标状态谓词不同的数量
def heu(self, state, posgoals, neggoals):
    h = 0
    for i in posgoals:
        if i not in state:
            h += 1
    for i in neggoals:
        if i in state:
            h += 1
    return h
 def solve(self, domain, problem):
 # 用启发式函数搜索部分
    visited = [state]
    bounder = [state, None]
```

```python
        while bounder:
            state = bounder.pop(0)
            plan = bounder.pop(0)
            h_min = 999
            for act in ground_actions:
                if self.applicable(state,
                    act.positive_preconditions,
                    act.negative_preconditions):
                     newstate = self.apply(state, act.add_effects,
                        act.del_effects)
                     if newstate not in visited:
                         flag = 0
                         if self.heu(newstate, posgoals, neggoals)
                             <= h_min:
                             flag = 1
                         if self.applicable(newstate, posgoals,
                             neggoals):
                             full_plan = [act]
                             while plan:
                                 act, plan = plan
                                 full_plan.insert(0, act)
                             return full_plan
                         if flag == 1:
                             visited.append(newstate)
                             bounder.append(newstate)
                             bounder.append((act, plan))
        return None
```

3. Explain any ideas you use to speed up the implementation. (10 points)
在求解过程中, 扩展 action 的时候, 首先判断一下该 action 的前提条件是怎样的, 如果这个 action 的前提条件不是当前 state 的子集, 那么就直接剪掉它; 还有就是建立了 `visited` 和 `bounder` 数组，将访问过的状态和该状态对应的可采纳的 action 全部存进去, 这样就不会出现在一些状态之间来回循环的情况; 并且再次调用的时候可以直接利用这两个数组所存的内容,

不需要再搜索一遍可能的 action, 这样节省了大量时间.

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

   5 个测例的输出如下, 包含运行时间和得到的解:

```
D:\Pycharm\PyCharm 2020.2.2\pddl-parser-master>python -B planner.py test0_domain.p
ddl test0_problem.pddl
Time: 0.000997781753540039s
plan:
action: move
  parameters: ('npc', 'town', 'field')
  positive_preconditions: [['at', 'npc', 'town'], ['border', 'town', 'field']]
  negative_preconditions: [['guarded', 'field']]
  add_effects: [['at', 'npc', 'field']]
  del_effects: [['at', 'npc', 'town']]

action: move
  parameters: ('npc', 'field', 'castle')
  positive_preconditions: [['at', 'npc', 'field'], ['border', 'field', 'castle']]
  negative_preconditions: [['guarded', 'castle']]
  add_effects: [['at', 'npc', 'castle']]
  del_effects: [['at', 'npc', 'field']]
```

Figure 1: Test Case0

```
D:\Pycharm\PyCharm 2020.2.2\pddl-parser-master>python -B planner.py test1_domain.p
ddl test1_problem.pddl
Time: 0.0s
plan:
action: move
  parameters: ('npc', 'town', 'tunnel')
  positive_preconditions: [['at', 'npc', 'town'], ['border', 'town', 'tunnel']]
  negative_preconditions: [['guarded', 'tunnel']]
  add_effects: [['at', 'npc', 'tunnel']]
  del_effects: [['at', 'npc', 'town']]

action: move
  parameters: ('npc', 'tunnel', 'river')
  positive_preconditions: [['at', 'npc', 'tunnel'], ['border', 'tunnel', 'river']]

  negative_preconditions: [['guarded', 'river']]
  add_effects: [['at', 'npc', 'river']]
  del_effects: [['at', 'npc', 'tunnel']]

action: move
  parameters: ('npc', 'river', 'castle')
  positive_preconditions: [['at', 'npc', 'river'], ['border', 'river', 'castle']]
  negative_preconditions: [['guarded', 'castle']]
  add_effects: [['at', 'npc', 'castle']]
  del_effects: [['at', 'npc', 'river']]
```

Figure 2: Test Case1

```
D:\Pycharm\PyCharm 2020.2.2\pddl-parser-master>python -B planner.py test2_domain.p
ddl test2_problem.pddl
Time: 0.004984617233276367s
plan:
action: move
  parameters: ('npc', 'town', 'field')
  positive_preconditions: [['at', 'npc', 'town'], ['border', 'town', 'field']]
  negative_preconditions: [['guarded', 'field']]
  add_effects: [['at', 'npc', 'field']]
  del_effects: [['at', 'npc', 'town']]

action: attack
  parameters: ('npc', 'ogre', 'field', 'river')
  positive_preconditions: [['at', 'npc', 'field'], ['at', 'ogre', 'river'], ['bord
er', 'field', 'river'], ['guarded', 'river']]
  negative_preconditions: []
  add_effects: []
  del_effects: [['at', 'ogre', 'river'], ['guarded', 'river']]

action: move
  parameters: ('npc', 'field', 'river')
  positive_preconditions: [['at', 'npc', 'field'], ['border', 'field', 'river']]
  negative_preconditions: [['guarded', 'river']]
  add_effects: [['at', 'npc', 'river']]
  del_effects: [['at', 'npc', 'field']]

action: attack
  parameters: ('npc', 'dragon', 'river', 'cave')
  positive_preconditions: [['at', 'npc', 'river'], ['at', 'dragon', 'cave'], ['bor
der', 'river', 'cave'], ['guarded', 'cave']]
  negative_preconditions: []
  add_effects: []
  del_effects: [['at', 'dragon', 'cave'], ['guarded', 'cave']]

action: open
  parameters: ('npc', 'box1', 'river')
  positive_preconditions: [['at', 'npc', 'river'], ['at', 'box1', 'river']]
  negative_preconditions: [['open', 'box1']]
  add_effects: [['open', 'box1']]
  del_effects: []

action: collect-fire
  parameters: ('npc', 'box1', 'river', 'reddust')
  positive_preconditions: [['at', 'npc', 'river'], ['at', 'box1', 'river'], ['open
', 'box1'], ['fire', 'reddust'], ['in', 'reddust', 'box1']]
  negative_preconditions: [['empty', 'box1']]
  add_effects: [['empty', 'box1'], ['has-fire', 'npc']]
  del_effects: []
```

```
action: move
  parameters: ('npc', 'river', 'cave')
  positive_preconditions: [['at', 'npc', 'river'], ['border', 'river', 'cave']]
  negative_preconditions: [['guarded', 'cave']]
  add_effects: [['at', 'npc', 'cave']]
  del_effects: [['at', 'npc', 'river']]

action: open
  parameters: ('npc', 'box2', 'cave')
  positive_preconditions: [['at', 'npc', 'cave'], ['at', 'box2', 'cave']]
  negative_preconditions: [['open', 'box2']]
  add_effects: [['open', 'box2']]
  del_effects: []

action: collect-earth
  parameters: ('npc', 'box2', 'cave', 'browndust')
  positive_preconditions: [['at', 'npc', 'cave'], ['at', 'box2', 'cave'], ['open',
 'box2'], ['earth', 'browndust'], ['in', 'browndust', 'box2']]
  negative_preconditions: [['empty', 'box2']]
  add_effects: [['empty', 'box2'], ['has-earth', 'npc']]
  del_effects: []

action: build-fireball
  parameters: ('npc',)
  positive_preconditions: [['has-fire', 'npc'], ['has-earth', 'npc']]
  negative_preconditions: []
  add_effects: [['has-fireball', 'npc']]
  del_effects: [['has-fire', 'npc'], ['has-earth', 'npc']]
```

Figure 3: Test Case2

```
D:\Pycharm\PyCharm 2020.2.2\pddl-parser-master>python -B planner.py test3_domain.p
ddl test3_problem.pddl
Time: 0.0009980201721191406s
plan:
action: unstack
  parameters: ('b', 'a', 'x')
  positive_preconditions: [['block', 'b'], ['block', 'a'], ['table', 'x'], ['on',
'a', 'x'], ['clear', 'b'], ['on', 'b', 'a']]
  negative_preconditions: []
  add_effects: [['on', 'b', 'x'], ['clear', 'a']]
  del_effects: [['on', 'b', 'a']]

action: move
  parameters: ('a', 'x', 'y')
  positive_preconditions: [['block', 'a'], ['table', 'x'], ['table', 'y'], ['on',
'a', 'x'], ['clear', 'a']]
  negative_preconditions: [['on', 'a', 'y']]
  add_effects: [['on', 'a', 'y']]
  del_effects: [['on', 'a', 'x']]

action: move
  parameters: ('b', 'x', 'y')
  positive_preconditions: [['block', 'b'], ['table', 'x'], ['table', 'y'], ['on',
'b', 'x'], ['clear', 'b']]
  negative_preconditions: [['on', 'b', 'y']]
  add_effects: [['on', 'b', 'y']]
  del_effects: [['on', 'b', 'x']]

action: stack
  parameters: ('a', 'b', 'y')
  positive_preconditions: [['block', 'a'], ['block', 'b'], ['table', 'y'], ['clear
', 'a'], ['clear', 'b'], ['on', 'a', 'y'], ['on', 'b', 'y']]
  negative_preconditions: []
  add_effects: [['on', 'a', 'b']]
  del_effects: [['on', 'a', 'y'], ['clear', 'b']]
```

Figure 4: Test Case3

```
D:\Pycharm\PyCharm 2020.2.2\pddl-parser-master>python -B planner.py test4_domain.p
ddl test4_problem.pddl
Time: 0.001953601837158203s
plan:
action: unstack
  parameters: ('b', 'a', 't1', 't3')
  positive_preconditions: [['block', 'b'], ['block', 'a'], ['table', 't1'], ['tabl
e', 't3'], ['on', 'a', 't1'], ['on', 'b', 'a'], ['clear', 'b'], ['clear', 't3']]
  negative_preconditions: []
  add_effects: [['on', 'b', 't3'], ['clear', 'a'], ['clear', 'b']]
  del_effects: [['on', 'b', 'a'], ['clear', 't3']]

action: stack
  parameters: ('a', 't1', 'b', 't3')
  positive_preconditions: [['block', 'a'], ['block', 'b'], ['table', 't1'], ['tabl
e', 't3'], ['clear', 'a'], ['clear', 'b'], ['on', 'a', 't1'], ['on', 'b', 't3']]
  negative_preconditions: []
  add_effects: [['on', 'a', 'b'], ['clear', 't1']]
  del_effects: [['on', 'a', 't1'], ['clear', 'b']]
```

Figure 5: Test Case4

可以看到, 这 5 个测例里最难的应该是测例 2, 需要 10 个步骤才能到达目标; 而且耗费时间也是最长的. 其余 4 个测例可以说差别很小, 尤其除了测例 4 的运行时间是 10^{-3} 数量级外，剩下的三个都是 10^{-4} 数量级, 可以说非常小了. 经检查, 输出的步骤里没有冗余的动作, 而且得到的结果也是最佳的 (正确且步骤最少). 这证明我们的 `STRIPS planner` 能在较短时间内跑出正确的结果, 实现的还算比较成功.

2 Variable Elimination

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='...' $\wedge$ 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)
   将概率图中所有变量分为三类, 分别是 evidence(已知变量),remaining(其他变量) 和 query(查询变量). 变量消除法的思想十分简单: 首先将所有的 evidence 直接代入每个节点的条件概率表 (CPT) 中, 这个 CPT 里面的变量也就被消除了一部分. 其次按照一定的顺序依次消除所有的 remaining, 初始化时我们根据代入 evidence 后的条件概率可以获得若干 factor, 这些 factor 里的变量显然不是 remaining 就是 query. 消除某个变量很简单, 就是将所有带该变量的 factor 取出来然后对于所有情况进行求和, 再删去原先的这些 factor 并创建新的 factor 即可, 新的 factor 内的变量列表就是原有的变量列表合并后去掉了被消除的变量的结果. 最后剩下的 factor 就是 query 的所有情况的概率, 这就是我们要的最后的结果. 至此，变量消除算法完成.

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')

这一部分内容在之前的实验已经实现, 就不在这里放代码了,Appendix 一栏里有本次实验整个 VE 算法的代码, 这里只放结果:

```
p1 = P(Mortality='True' & CTScanResult='Ischemic Stroke' | PatientAge='31-65' )
RESULT:
['C', 'N']
Name = f['C', 'N']
 vars ['C', 'N']
   key: 00 val : 0.283605
   key: 01 val : 0.41639499999999996
   key: 10 val : 0.17587499999999998
   key: 11 val : 0.12412500000000001
```

Figure 6: p1

```
p2 = P(Disability='Moderate' & CTScanResult='Hemmorraghic Stroke' | PatientAge='65+'& MRIScanResult='Hemmorraghic Stroke')
RESULT:
['D', 'C']
Name = f['C', 'D']
 vars ['C', 'D']
   key: 00 val : 0.16800000000000004
   key: 01 val : 0.20300000000000004
   key: 02 val : 0.329
   key: 10 val : 0.057
   key: 11 val : 0.057
   key: 12 val : 0.18600000000000005
```

Figure 7: p2

```
p3 = P(StrokeType='Hemmorraghic Stroke' | PatientAge='65+' & CTScanResult='Hemmorraghic Stroke' & MRIScanResult='Ischemic Stroke')
RESULT:
['S']
['S']
Name = f['S']
 vars ['S']
   key: 0 val : 0.5000000000000001
   key: 1 val : 0.4
   key: 2 val : 0.10000000000000002
```

Figure 8: p3

```
p4 = P(Anticoagulants='Used' | PatientAge='31-65')
RESULT:
['A']
Name = f['A']
 vars ['A']
   key: 0 val : 0.5
   key: 1 val : 0.49999999999999994
```

Figure 9: p4

```
p5 = P(Disability='Negligible')
RESULT:
Name = f['D']
 vars ['D']
   key: 0 val : 0.38977
   key: 1 val : 0.292515
   key: 2 val : 0.317715
```

Figure 10: p5

3. Implement an algorithm to select a good order of variable elimination. (10 points)
   下面的两个函数是选择较好的变量消除的顺序. 其中 `chooseElimOrderInMinEdge` 是选取度最小的那个变量;`chooseElimOrderInBestFill` 则是在删去变量后添加边的最优方式. 经过多次测试我们发现第一个执行效率更高, 所以我们在后面就选取了 `chooseElimOrderInMinEdge` 这个函数来测试性能.

```python
def chooseElimOrderInMinEdge(preorder, evidence, graph):
    for ie in evidence:
        for vertex in graph.vertexList:
            if vertex.name == trans[ie]:
                graph.remove(vertex)
    aftorder = []
    for i in range(len(preorder)):
        min = 999
        minNode = 'C'
        minVertex = Cv
        for vertex in graph.vertexList:
            # print(vertex.name)
            if vertex.name not in [trans[i] for i in preorder]:
                continue
            else:
                # print(vertex.name, ' - ', len(vertex.neighbor),
                    vertex.neighbor)
                if len(vertex.neighbor) < min:
                    min = len(vertex.neighbor)
                    minNode =
                        list(trans.keys())[list(trans.values()).
```

```
                    index(vertex.name)]
                    minVertex = vertex
        aftorder.append(minNode)
        # print(minNode)
        graph.remove(minVertex)
        preorder.remove(minNode)
    return aftorder


def chooseElimOrderInBestFill(preorder, evidence, graph):
        for ie in evidence:
            for vertex in graph.vertexList:
                if vertex.name == trans[ie]:
                    graph.remove(vertex)
        aftorder = []
        for i in range(len(preorder)):
            min = 999
            minNode = 'C'
            minVertex = Cv
            for vertex in graph.vertexList:
                if vertex.name not in [trans[i] for i in preorder]:
                    continue
                else:
                    if graph.newEdge(vertex) < min:
                        print(vertex.name, ' - ',
                            graph.newEdge(vertex))
                        min = graph.newEdge(vertex)
                        minNode =
                            list(trans.keys())[list(trans.values()).
                        index(vertex.name)]
                        minVertex = vertex
            aftorder.append(minNode)
            print(minNode)
            graph.remove(minVertex)
```

```python
            preorder.remove(minNode)
        return aftorder
```

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 width, 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)

   `Total time` 一栏里填写的时间为样例跑 1000 次所用的平均时间.

Test case	Elimination order	Elimination width	Total time
p4	C,M,D,S,N(Good)	3	0.267ms
p4	N,C,M,S,D	3	0.313ms
p4	M,C,S,N,D	3	0.395ms
p4	N,S,C,M,D	4	0.570ms
p4	C,S,M,N,D	4	0.627ms
p4	S,M,N,C,D	5	0.976ms
p5	P,A,C,M,N,S(Good)	3	0.340ms
p5	M,C,N,P,S,A	3	0.359ms
p5	A,P,M,N,S,C	3	0.387ms
p5	P,M,S,A,C,N	4	0.737ms
p5	N,S,P,A,M,C	5	1.455ms
p5	S,C,P,A,M,N	6	2.632ms

   这张表格标有 Good 的那一行是调用了我们自己写的函数 `chooseElimOrderInMinEdge` 得到的结果 (即选择一个较好的顺序做消除). 每个测例按照时间从小到大进行排列. 观察表格我们可以很容易得到如下结论: 运行时间与 `Elimination width` 有关,`Elimination width` 越大, 运行时间越长; 当 `Elimination width` 相等时可以看到: 不论消除顺序怎样, 运行时间都很接近. 当然 `Elimination width` 是由 `Elimination order` 决定的.

3 Appendix

这里将全部代码列出:

3.1 STRIPS planner

首先是修改后的 PDDL 文件:

test0_domain.pddl

```
(define (domain magic-world)
   (:requirements :strips :typing)
   (:types player location monster element chest)

   (:action move
      :parameters (?p - player ?l1 - location ?l2 - location)
      :precondition (and (at ?p ?l1) (border ?l1 ?l2) (not (guarded ?l2)
         ))
      :effect (and (at ?p ?l2) (not (at ?p ?l1)))
   )
)
```

test0_problem.pddl

```
(define (problem move-to-castle)
   (:domain magic-world)

   (:objects
      npc - player
      town field castle - location
   )

   (:init
      (border town field)
      (border field castle)

      (at npc town)
   )

   (:goal (and (at npc castle)))
)
```

test1_domain.pddl

```
(define (domain magic-world)
   (:requirements :strips :typing)
   (:types player location monster element chest)

   (:action move
      :parameters (?p - player ?l1 - location ?l2 - location)
      :precondition (and (at ?p ?l1) (border ?l1 ?l2) (not (guarded ?l2)
         ))
      :effect (and (at ?p ?l2) (not (at ?p ?l1)))
   )
)
```

test1_problem.pddl

```
(define (problem sneak-past-dragon-to-castle)
   (:domain magic-world)

   (:objects
      npc - player
      dragon - monster
      town field castle tunnel river - location
   )
```

```
  (:init
    (border town field)
    (border town tunnel)
    (border field castle)
    (border tunnel river)
    (border river castle)

    (at npc town)
    (at dragon field)
    (guarded field)
  )

  (:goal (and (at npc castle)))
)
```

test2_domain.pddl

```
(define (domain magic-world)
  (:requirements :strips :typing)
  (:types player location monster element chest)

  (:action move
    :parameters (?p - player ?l1 - location ?l2 - location)
    :precondition (and (at ?p ?l1) (border ?l1 ?l2) (not (guarded ?l2)
      ))
    :effect (and (at ?p ?l2) (not (at ?p ?l1)))
  )

  (:action attack
    :parameters (?p - player ?m - monster ?l1 - location ?l2 -
      location)
    :precondition (and (at ?p ?l1) (at ?m ?l2) (border ?l1 ?l2) (
      guarded ?l2))
    :effect (and (not (at ?m ?l2)) (not (guarded ?l2)))
  )

  (:action open
    :parameters (?p - player ?c - chest ?l1 - location)
    :precondition (and (at ?p ?l1) (at ?c ?l1) (not (open ?c)))
    :effect (and (open ?c))
  )

  (:action collect-fire
    :parameters (?p - player ?c - chest ?l1 - location ?e - element)
    :precondition (and (at ?p ?l1) (at ?c ?l1) (open ?c) (fire ?e) (in
       ?e ?c) (not (empty ?c)))
    :effect (and (empty ?c) (has-fire ?p))
  )

  (:action collect-earth
    :parameters (?p - player ?c - chest ?l1 - location ?e - element)
    :precondition (and (at ?p ?l1) (at ?c ?l1) (open ?c) (earth ?e) (
      in ?e ?c) (not (empty ?c)))
    :effect (and (empty ?c) (has-earth ?p))
  )

  (:action build-fireball
    :parameters (?p - player)
    :precondition (and (has-fire ?p) (has-earth ?p))
    :effect (and (has-fireball ?p) (not (has-fire ?p)) (not (has-earth
       ?p)))
  )
```

)

test2_problem.pddl

```
(define (problem fireball)
   (:domain magic-world)

   (:objects
      npc - player
      ogre dragon - monster
      town field river cave - location
      box1 box2 - chest
      reddust browndust - element
   )

   (:init
      (border town field)
      (border field town)
      (border field river)
      (border river field)
      (border river cave)
      (border cave river)

      (at npc town)
      (at ogre river)
      (at dragon cave)
      (guarded river)
      (guarded cave)

      (at box1 river)
      (at box2 cave)

      (fire reddust)
      (in reddust box1)

      (earth browndust)
      (in browndust box2)
   )

   (:goal (and (has-fireball npc)))
)
```

test3_domain.pddl

```
(define (domain blocksworld)
  (:requirements :strips :typing)
  (:types block table)
  (:action move
     :parameters (?b - block ?t1 - table ?t2 - table)
     :precondition (and (block ?b) (table ?t1) (table ?t2) (on ?b ?t1) (
        not (on ?b ?t2)) (clear ?b))
     :effect (and (on ?b ?t2) (not (on ?b ?t1)))
     )
  (:action stack
     :parameters (?a - block ?b - block ?t1 - table)
     :precondition (and (block ?a) (block ?b) (table ?t1) (clear ?a) (
        clear ?b) (on ?a ?t1) (on ?b ?t1))
     :effect (and (on ?a ?b) (not (on ?a ?t1)) (not (clear ?b)))
     )
  (:action unstack
     :parameters (?a - block ?b - block ?t1 - table)
     :precondition (and (block ?a) (block ?b) (table ?t1) (on ?b ?t1) (
        clear ?a) (on ?a ?b))
```

```
      :effect (and (on ?a ?t1) (not (on ?a ?b)) (clear ?b))
      )
)
```

test3_problem.pddl

```
(define (problem
   stack-blocks-stacked-ba-from-tablex-to-stacked-ab-tabley)
  (:domain blocksworld)
  (:objects
    a b - block
    x y - table)
  (:init (block a) (block b) (table x) (table y)
         (on a x) (on b a) (clear b))
  (:goal (and (on b y) (on a b) (clear a) (not (clear b))))
)
```

test4_domain.pddl

```
(define (domain blocksworld)
  (:requirements :strips :typing)
  (:types block table)
  (:action move
      :parameters (?b - block ?x - table ?y - table)
      :precondition (and (block ?b) (table ?x) (table ?y) (on ?b ?x) (
         clear ?b) (clear ?y))
      :effect (and (not (on ?b ?x)) (on ?b ?y) (clear ?x) (not (clear ?y)
         ))
      )
  (:action stack
      :parameters (?a - block ?x - table  ?b - block  ?y - table)
      :precondition (and (block ?a) (block ?b) (table ?x) (table ?y) (
         clear ?a) (clear ?b) (on ?a ?x) (on ?b ?y))
      :effect (and (on ?a ?b) (not (on ?a ?x)) (not (clear ?b)) (clear ?x
         ))
      )
  (:action unstack
      :parameters (?a - block ?b - block ?x - table ?y - table)
      :precondition (and (block ?a) (block ?b) (table ?x) (table ?y) (on
         ?b ?x) (on ?a ?b) (clear ?a) (clear ?y))
      :effect (and (on ?a ?y) (not (on ?a ?b)) (clear ?b) (clear ?a) (not
          (clear ?y)))
      )
)
```

test4_problem.pddl

```
(define (problem
   stack-blocks-stacked-ba-from-table1-to-stacked-ab-table3-
onepilepertable)
  (:domain blocksworld)
  (:objects
      a b - block
      t1 t2 t3 - table
  )
  (:init (block a) (block b) (table t1) (table t2) (table t3)
         (on a t1) (on b a) (clear b) (clear t2) (clear t3))
  (:goal (and (on a b) (on b t3)))
)
```

实现 STRIPS planner 的代码文件 planner.py 如下:

```
from PDDL import PDDL_Parser

class Planner:
# heuristic
    def heu(self, state, posgoals, neggoals):
        h = 0
        for i in posgoals:
            if i not in state:
                h += 1
        for i in neggoals:
            if i in state:
                h += 1
        return h
# Solve
    def solve(self, domain, problem):
        # Parser
        parser = PDDL_Parser()
        parser.domainparser(domain)
        parser.problemparser(problem)
        # Parsed data
        state = parser.state
        posgoals = parser.positive_goals
        neggoals = parser.negative_goals
        if self.applicable(state, posgoals, neggoals):
            return []
        # Grounding process
        ground_actions = []
        for action in parser.actions:
            for act in action.groundify(parser.objects):
                ground_actions.append(act)
        # Search
        visited = [state]
        bounder = [state, None]
        while bounder:
            state = bounder.pop(0)
            plan = bounder.pop(0)
            h_min = 999
            for act in ground_actions:
                if self.applicable(state, act.positive_preconditions,
                   act.negative_preconditions):
                    newstate = self.apply(state, act.add_effects,
                        act.del_effects)
                    if newstate not in visited:
                        flag = 0
                        if self.heu(newstate, posgoals, neggoals) <=
                           h_min:
                            flag = 1
                        if self.applicable(newstate, posgoals,
                           neggoals):
                            full_plan = [act]
                            while plan:
                                act, plan = plan
                                full_plan.insert(0, act)
                            return full_plan
                        if flag == 1:
                            visited.append(newstate)
                            bounder.append(newstate)
                            bounder.append((act, plan))
        return None
# Judge
    def applicable(self, state, positive, negative):
```

```python
        for i in positive:
            if i not in state:
                return False
        for i in negative:
            if i in state:
                return False
        return True
# Apply
    def apply(self, state, positive, negative):
        newstate = []
        for i in state:
            if i not in negative:
                newstate.append(i)
        for i in positive:
            if i not in newstate:
              newstate.append(i)
        return newstate
# Main
if __name__ == '__main__':
    import sys, time
    start_time = time.time()
    domain = sys.argv[1]
    problem = sys.argv[2]
    planner = Planner()
    plan = planner.solve(domain, problem)
    print('Time: ' + str(time.time() - start_time) + 's')
    if plan:
        print('plan:')
        for act in plan:
            print(act)
    else:
        print('No plan was found!')
```

PDDL parser 的实现如下, 包含两个文件:`PDDL.py`,`action.py`, 参考了网上的代码自己做了些修改:

PDDL.py

```python
import re
from action import Action

class PDDL_Parser:
    SUPPORTED_REQUIREMENTS = [':strips', ':negative-preconditions',
        ':typing']

    def check(self, filename):
        with open(filename,'r') as f:
            # Remove single line comments
            str = re.sub(r';.*$', '', f.read(),
                flags=re.MULTILINE).lower()
        # Tokenize
        stack = []
        list = []
        for t in re.findall(r'[()]|[^\s()]+', str):
            if t == '(':
                stack.append(list)
                list = []
            elif t == ')':
                if stack:
                    l = list
                    list = stack.pop()
                    list.append(l)
                else:
```

```
                raise Exception('Missing open parentheses')
            else:
                list.append(t)
        if stack:
            raise Exception('Missing close parentheses')
        if len(list) != 1:
            raise Exception('Malformed expression')
        return list[0]

    def domainparser(self, domain_filename):
        tokens = self.check(domain_filename)
        if type(tokens) is list and tokens.pop(0) == 'define':
            self.domain_name = 'unknown'
            self.requirements = []
            self.types = []
            self.actions = []
            self.predicates = {}
            while tokens:
                group = tokens.pop(0)
                t = group.pop(0)
                if   t == 'domain':
                    self.domain_name = group[0]
                elif t == ':requirements':
                    for req in group:
                        if not req in self.SUPPORTED_REQUIREMENTS:
                            raise Exception('Requirement ' + req + '
                                not supported')
                    self.requirements = group
                elif t == ':predicates':
                    self.preparser(group)
                elif t == ':types':
                    self.types = group
                elif t == ':action':
                    self.actparser(group)
                else: print(str(t) + ' is not recognized in domain')
        else:
            raise Exception('File ' + domain_filename + ' does not
              match domain pattern')

    def preparser(self, group):
        for pred in group:
            predicate_name = pred.pop(0)
            if predicate_name in self.predicates:
                raise Exception('Predicate ' + predicate_name + '
                  redefined')
            arguments = {}
            untyped_variables = []
            while pred:
                t = pred.pop(0)
                if t == '-':
                    if not untyped_variables:
                        raise Exception('Unexpected hyphen in
                          predicates')
                    type = pred.pop(0)
                    while untyped_variables:
                        arguments[untyped_variables.pop(0)] = type
                else:
                    untyped_variables.append(t)
            while untyped_variables:
                arguments[untyped_variables.pop(0)] = 'object'
            self.predicates[predicate_name] = arguments
```

```python
def actparser(self, group):
    name = group.pop(0)
    if not type(name) is str:
        raise Exception('Action without name definition')
    for act in self.actions:
        if act.name == name:
            raise Exception('Action ' + name + ' redefined')
    parameters = []
    positive_preconditions = []
    negative_preconditions = []
    add_effects = []
    del_effects = []
    while group:
        t = group.pop(0)
        if t == ':parameters':
            if not type(group) is list:
                raise Exception('Error with ' + name + '
                    parameters')
            parameters = []
            untyped_parameters = []
            p = group.pop(0)
            while p:
                t = p.pop(0)
                if t == '-':
                    if not untyped_parameters:
                        raise Exception('Unexpected hyphen in ' +
                            name + ' parameters')
                    ptype = p.pop(0)
                    while untyped_parameters:
                        parameters.append([untyped_parameters.pop(0),
                            ptype])
                else:
                    untyped_parameters.append(t)
            while untyped_parameters:
                parameters.append([untyped_parameters.pop(0),
                    'object'])
        elif t == ':precondition':
            self.presplit(group.pop(0), positive_preconditions,
                negative_preconditions, name, ' preconditions')
        elif t == ':effect':
            self.presplit(group.pop(0), add_effects, del_effects,
                name, ' effects')
        else: print(str(t) + ' is not recognized in action')
    self.actions.append(Action(name, parameters,
        positive_preconditions, negative_preconditions, add_effects,
        del_effects))

def problemparser(self, problem_filename):
    tokens = self.check(problem_filename)
    if type(tokens) is list and tokens.pop(0) == 'define':
        self.problem_name = 'unknown'
        self.objects = dict()
        self.state = []
        self.positive_goals = []
        self.negative_goals = []
        while tokens:
            group = tokens.pop(0)
            t = group[0]
            if   t == 'problem':
                self.problem_name = group[-1]
            elif t == ':domain':
                if self.domain_name != group[-1]:
```

```
                        raise Exception('Different domain specified in
                           problem file')
                elif t == ':requirements':
                    pass
                    '''Ignore requirements in problem, parse them in
                       the domain'''
                elif t == ':objects':
                    group.pop(0)
                    object_list = []
                    while group:
                        if group[0] == '-':
                            group.pop(0)
                            self.objects[group.pop(0)] = object_list
                            object_list = []
                        else:
                            object_list.append(group.pop(0))
                    if object_list:
                        if not 'object' in self.objects:
                            self.objects['object'] = []
                        self.objects['object'] += object_list
                elif t == ':init':
                    group.pop(0)
                    self.state = group
                elif t == ':goal':
                    self.presplit(group[1], self.positive_goals,
                       self.negative_goals, '', 'goals')
                else: print(str(t) + ' is not recognized in problem')
        else:
            raise Exception('File ' + problem_filename + ' does not
               match problem pattern')

        # split predicates
    def presplit(self, group, pos, neg, name, part):
        if not type(group) is list:
            raise Exception('Error with ' + name + part)
        if group[0] == 'and':
            group.pop(0)
        else:
            group = [group]
        for predicate in group:
            if predicate[0] == 'not':
                if len(predicate) != 2:
                    raise Exception('Unexpected not in ' + name + part)
                neg.append(predicate[-1])
            else:
                pos.append(predicate)
```

action.py

```
import itertools

class Action:
    # 初始化
    def __init__(self, name, parameters, positive_preconditions,
       negative_preconditions, add_effects, del_effects):
        self.name = name
        self.parameters = parameters
        self.positive_preconditions = positive_preconditions
        self.negative_preconditions = negative_preconditions
        self.add_effects = add_effects
        self.del_effects = del_effects
    # 打印信息
```

```
    def __str__(self):
        return 'action: ' + self.name + \
        '\n parameters: ' + str(self.parameters) + \
        '\n positive_preconditions: ' +
            str(self.positive_preconditions) + \
        '\n negative_preconditions: ' +
            str(self.negative_preconditions) + \
        '\n add_effects: ' + str(self.add_effects) + \
        '\n del_effects: ' + str(self.del_effects) + '\n'
    # 比较
    def __eq__(self, other):
        return self.__dict__ == other.__dict__
        # ground process
    def groundify(self, objects):
        if not self.parameters:
            yield self
            return
        type_map = []
        variables = []
        for var, type in self.parameters:
            type_map.append(objects[type])
            variables.append(var)
        for assignment in itertools.product(*type_map):
            positive_preconditions =
                self.replace(self.positive_preconditions, variables,
                assignment)
            negative_preconditions =
                self.replace(self.negative_preconditions, variables,
                assignment)
            add_effects = self.replace(self.add_effects, variables,
                assignment)
            del_effects = self.replace(self.del_effects, variables,
                assignment)
            yield Action(self.name, assignment, positive_preconditions,
                negative_preconditions, add_effects, del_effects)

    def replace(self, group, variables, assignment):
        g = []
        for pred in group:
            pred = list(pred)
            iv = 0
            for v in variables:
                while v in pred:
                    pred[pred.index(v)] = assignment[iv]
                iv += 1
            g.append(pred)
        return g
```

3.2 Variable Elimination

变量消除算法的实现:

VE.py

```
#-*- coding:utf-8 -*-
import timeit,sys
import datetime
import numpy as np

class Vertex:
```

```
    def __init__(self, name, neighbor):
        self.name = name
        self.neighbor = neighbor
        self.store = neighbor

    def recover(self):
        self.neighbor = self.store

class Graph:
    def __init__(self, vertexList):
        self.name = "Never Give Up"
        self.vertexList = vertexList
        self.store = vertexList

    def degree(self, toBeElim):
        return len(self.vertexList[toBeElim].neighbor)

    def newEdge(self, toBeElim):
        neighbors = []
        for ver in self.vertexList:
            if ver == toBeElim:
                neighbors = ver.neighbor
        newEdge = 0
        print(toBeElim.name, ' ', neighbors)
        for verName in neighbors:
            for iver in self.vertexList:
                if verName == iver.name:
                    ver = iver
            for iver in neighbors:
                if (not verName == iver) and (iver not in ver.neighbor):
                    newEdge += 1
        return newEdge / 2

    def remove(self, toBeElim):
        name = toBeElim.name
        neighbors = []
        for ver in self.vertexList:
            if ver == toBeElim:
                neighbors = ver.neighbor
        newEdge = 0
        for verName in neighbors:
            for iver in self.vertexList:
                if verName == iver.name:
                    ver = iver
            for iver in neighbors:
                if (not verName == iver) and (iver not in ver.neighbor):
                    ver.neighbor.append(iver)
                    newEdge += 1
        for ver in self.vertexList:
            if name in ver.neighbor:
                ver.neighbor.remove(name)
        self.vertexList.remove(toBeElim)

    def recover(self):
        self.vertexList = self.store
        # for vertex in self.vertexList:
        #     vertex.recover()


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

```python
        for evidence in evidenceList:
            # Your code here
            # 把evidence全部实例化
            for factor in factorList:
                if evidence in factor.varList:
                    if evidence in factor.varList:
                        if len(factor.varList) > 1:
                            factorList.append(factor.restrict(evidence,
                                evidenceList[evidence]))
                        factorList.remove(factor)

            # Your code end

        for ivariable in orderedListOfHiddenVariables:
            int_s = datetime.datetime.now().microsecond
            # Your code here
            # 变量删除
            toBeEliminate = list(filter(lambda afactor : ivariable in
                afactor.varList, factorList))
            new_var = toBeEliminate[0]
            for e in toBeEliminate:
                for i in factorList:
                    if i.name == e.name:
                        factorList.remove(i)

                if not 0 == toBeEliminate.index(e):
                    new_var = new_var.multiply(e)

            new_var = new_var.sumout(ivariable)
            factorList.append(new_var)
            int_e = datetime.datetime.now().microsecond
            print(ivariable, ' -> ', int_e - int_s)
            # Your code end


        print("RESULT:")
        res = factorList[0]
        for factor in factorList[1:]:
            print(factor.varList)
            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):
```

```python
        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 = [var for var in self.varList]
        new_cpt = {}

        # 存储相同变量在两个varList中的位置
        idx1 = []
        idx2 = []

        for var2 in factor.varList:
            if var2 in newList:
                idx1.append(self.varList.index(var2))
                idx2.append(factor.varList.index(var2))
            else:
                newList.append(var2)
                # 把factor中有而self中没有的变量存入newList
                # 这样newList就包含两个node的全部变量
        # print(idx1,'|',idx2)
        for k1, v1 in self.cpt.items():
            for k2, v2 in factor.cpt.items():   # v1,v2是两个概率
                flag = True
                # 用于判断两个items中相同变量的正负性是否一致
                # 一致则说明找到用于相乘的两个变量items
                for i in range(len(idx1)):
                    if k1[idx1[i]] != k2[idx2[i]]:
                    # 存在同一变量在两边正负性相反的情况
                        flag = False
                        break
                if flag:
                    new_key = k1
                    for i in range(len(k2)):
                        if i in idx2:
                            continue
                        new_key += k2[i]     # 对应newList中的各个变量
                    new_cpt[new_key] = v1 * v2       # 概率相乘

        #Your code end
        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_var_list = [var for var in self.varList]
        new_var_list.remove(variable)        # 删去需要累加的变量
        new_cpt = {}

        idx = self.varList.index(variable)  # 做累加的变量的位置
```

```
        for k, v in self.cpt.items():
            if k[:idx] + k[idx + 1:] not in new_cpt.keys():
            # 还没记录的变量组合：创建和赋值
                new_cpt[k[:idx] + k[idx + 1:]] = v
            else:
            # 已经记录的变量组合：累加
                new_cpt[k[:idx] + k[idx + 1:]] += v

        #Your code end
        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"""
        # 也就是具体化一个变量
        #Your code here

        new_var_list = [i for i in self.varList]
        new_var_list.remove(variable)              # 删去需要具体化的变量
        new_cpt = {}

        idx = self.varList.index(variable)         # 具体化的变量的位置

        # 例如，现在要将Pr(A,B,C)具体化为Pr(~a,B,C)
        # 则传入参数variable = 'a', value = 0
        '''对cpt中一项(["A","B","C"],'011': 0.19),
        经过变化后得到(["B","C"], '11' : 0.19)'''
        for k, v in self.cpt.items():
            if k[idx] == str(value):
                new_cpt[k[:idx] + k[idx + 1:]] = v

        #Your code end
        new_node = Node("f" + str(new_var_list), new_var_list)
        new_node.setCpt(new_cpt)
        return new_node

def chooseElimOrderInMinEdge(preorder, evidence, graph):
    for ie in evidence:
        for vertex in graph.vertexList:
            if vertex.name == trans[ie]:
                graph.remove(vertex)
    aftorder = []
    for i in range(len(preorder)):
        min = 999
        minNode = 'C'
        minVertex = Cv
        for vertex in graph.vertexList:
            # print(vertex.name)
            if vertex.name not in [trans[i] for i in preorder]: continue
            else:
                # print(vertex.name, ' - ', len(vertex.neighbor),
                    vertex.neighbor)
                if len(vertex.neighbor) < min:
                    min = len(vertex.neighbor)
                    minNode = list(trans.keys())[list(trans.values()).
                    index(vertex.name)]
                    minVertex = vertex
        aftorder.append(minNode)
        # print(minNode)
        graph.remove(minVertex)
```

```
        preorder.remove(minNode)
    return afterorder

def chooseElimOrderInBestFill(preorder, evidence, graph):
    for ie in evidence:
        for vertex in graph.vertexList:
            if vertex.name == trans[ie]:
                graph.remove(vertex)
    afterorder = []
    for i in range(len(preorder)):
        min = 999
        minNode = 'C'
        minVertex = Cv
        for vertex in graph.vertexList:
            if vertex.name not in [trans[i] for i in preorder]: continue
            else:
                if graph.newEdge(vertex) < min:
                    print(vertex.name, ' - ', graph.newEdge(vertex))
                    min = graph.newEdge(vertex)
                    minNode = list(trans.keys())[list(trans.values()).
                    index(vertex.name)]
                    minVertex = vertex
        afterorder.append(minNode)
        print(minNode)
        graph.remove(minVertex)
        preorder.remove(minNode)
    return afterorder

def chooseElimOrderInRandom(preorder, evidence):
    weight = {}
    afterorder = []
    for i in preorder:
        weight[i] = np.random.rand()
    weight = sorted(weight.items(), key = lambda kv:(kv[1], kv[0]))
    # print(weight)
    for i in weight:
        afterorder.append(i[0])
    # print(afterorder)
    return afterorder

def main():
    global trans
    trans = {'C': 'Cv', 'M': 'Mv', 'S': 'Sv', 'N': 'Nv', 'A': 'Av',
        'P': 'Pv', 'D': 'Dv'}
    # create nodes for Bayes Net
    P = Node("P", ["P"])                                # PatientAge
    C = Node("C", ["C"])                                # CTScanResult
    M = Node("M", ["M"])                                # MRIScanResult
    A = Node("A", ["A"])                                # Anticoagulants
    S = Node("S", ["S", "C", "M"])                      # StrokeType
    N = Node("N", ["N", "S", "A"])                      # Mortality
    D = Node("D", ["D", "S", "P"])                      # Disability

    # 建立邻接表
    global Pv                                   # PatientAge
    Pv = Vertex('Pv', ['Dv'])
    global Cv                                   # CTScanResult
    Cv = Vertex('Cv', ['Sv'])
    global Mv                                   # MRIScanResult
    Mv = Vertex('Mv', ['Sv'])
    global Av                                   # Anticoagulants
    Av = Vertex('Av', ['Nv'])
```

```
global Sv                              # StrokeType
Sv = Vertex('Sv', ['Cv', 'Mv', 'Nv', 'Dv'])
global Nv                              # Mortality
Nv = Vertex('Nv', ['Sv', 'Av'])
global Dv                         # Disability
Dv = Vertex('Dv', ['Sv', 'Pv'])

G = Graph([Pv, Av, Sv, Cv, Mv, Nv, Dv])

# Generate cpt for each node
P.setCpt({'0': 0.1, '1': 0.3, '2': 0.6})
C.setCpt({'0': 0.7, '1': 0.3})
M.setCpt({'0': 0.7, '1': 0.3})
A.setCpt({'0': 0.5, '1': 0.5})
S.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})
N.setCpt({'000': 0.56, '010': 0.58, '020': 0.05, '001': 0.28,
    '011': 0.99, '021': 0.10,'100': 0.44, '110': 0.42, '120': 0.95,
    '101': 0.72, '111': 0.01, '121': 0.90})
D.setCpt({'000': 0.80, '010': 0.70, '020': 0.90, '001': 0.60,
    '011': 0.50, '021': 0.40, '002': 0.30, '012': 0.20, '022':
    0.10,'100': 0.10, '110': 0.20, '120': 0.05, '101': 0.30, '111':
    0.40, '121': 0.30, '102': 0.40, '112': 0.20, '122': 0.10,'200':
    0.10, '210': 0.10, '220': 0.05, '201': 0.10, '211': 0.10, '221':
    0.30, '202': 0.30, '212': 0.60, '222': 0.80})


print("p1 = P(Mortality=' True' & CTScanResult=' Ischemic Stroke'
    | PatientAge=' 31-65' )")
VariableElimination.inference([P,C,M,A,S,N,D], ['N','C'],
    ['M','A','S','D'], {'P':1})

print("p2 = P(Disability=' Moderate' & CTScanResult=' Hemmorraghic
    Stroke' | PatientAge=' 65+' & MRIScanResult=' Hemmorraghic
    Stroke' )")
VariableElimination.inference([P,C,M,A,S,N,D], ['D','C'],
    ['A','S','N'], {'P':2,'M':1})

print("p3 = P(StrokeType=' Hemmorraghic Stroke' |
    PatientAge=' 65+' & CTScanResult=' Hemmorraghic Stroke' &
    MRIScanResult=' Ischemic Stroke' )")
VariableElimination.inference([P,C,M,A,S,N,D], ['S'],
    ['A','N','D'], {'P':2,'C':1,'M':0})

print("p4 = P(Anticoagulants=' Used' | PatientAge=' 31-65' )")
order4 = chooseElimOrderInBestFill(['C', 'M', 'S', 'N', 'D'],
    ['P'], G)
VariableElimination.inference([P,C,M,A,S,N,D], ['A'], order4,
    {'P':1})
# 恢复到处理前的状态
Pv = Vertex('Pv', ['Dv'])
Cv = Vertex('Cv', ['Sv'])
Mv = Vertex('Mv', ['Sv'])
Av = Vertex('Av', ['Nv'])
Sv = Vertex('Sv', ['Cv', 'Mv', 'Nv', 'Dv'])
Nv = Vertex('Nv', ['Sv', 'Av'])
Dv = Vertex('Dv', ['Sv', 'Pv'])
G = Graph([Pv, Av, Sv, Cv, Mv, Nv, Dv])

print("p5 = P(Disability=' Negligible' )")
order5 = chooseElimOrderInBestFill(['C', 'M', 'S', 'N', 'A', 'P'],
```

```
        [], G)
    VariableElimination.inference([P,C,M,A,S,N,D], ['D'], order5, {})
    # 恢复到处理前的状态
    Pv = Vertex('Pv', ['Dv'])
    Cv = Vertex('Cv', ['Sv'])
    Mv = Vertex('Mv', ['Sv'])
    Av = Vertex('Av', ['Nv'])
    Sv = Vertex('Sv', ['Cv', 'Mv', 'Nv', 'Dv'])
    Nv = Vertex('Nv', ['Sv', 'Av'])
    Dv = Vertex('Dv', ['Sv', 'Pv'])
    G = Graph([Pv, Av, Sv, Cv, Mv, Nv, Dv])

if __name__ == '__main__':
    main()
```