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

P03 Planning and Uncertainty

18340215 张天祎

2020 年 11 月 25 日

目录

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)

   - 可达性分析启发式函数的核心思想是考虑这样一个松弛问题：规划的所有行动都不考虑它们效果中的删除部分，每一层执行所有能执行的动作（默认不考虑已经执行过的任务，因为他们不会对知识库有影响），这样问题的知识库只会增加，不会减少。若在这样的松弛问题中都达不到目标，那该状态大概率无法到达目标，其启发式函数值较大；若能达到，再使用 Count-Actions 的思路，每次用行动层的效果覆盖上一层的新前提来尽可能模拟到达目标的过程，启发式函数值为所有层行动的总数。

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

```python
import re
import copy
from timeit import default_timer as timer

# 测试时使用的，实际上搜索树很浅
MAX_DEPTH = 50

def parser(file_name):
    str = open(file_name, 'r').read()

    # https://github.com/pucrs-automated-planning/pddl-parser/blob/master/PDDL.py
    stack = []
    list = []
    for t in re.findall(r'[()]|[^\s()]+', str):
        if t == '(':
            stack.append(list)
            list = []
```

```
18            elif t == ')':
19                if stack:
20                    l = list
21                    list = stack.pop()
22                    list.append(l)
23                else:
24                    raise Exception('Missing open parentheses')
25            else:
26                list.append(t)
27        return list[0]
28
29    class Action:
30        def __init__(self, init_list, types):
31            self.name = init_list[1]
32            self.parameters = {}
33
34            if types != ['AnyType']:
35                for i in range(0, len(init_list[3]), 3):
36                    if init_list[3][i+2] in types:
37                        self.parameters[init_list[3][i][1:]] = init_list[3][i +
                              2]
38            else:
39                for i in init_list[3]:
40                    self.parameters[i[1:]] = 'AnyType'
41            self.precondition = []
42            for i in init_list[5]:
43                if type(i) is list:
44                    if i[0] != 'not':
45                        self.precondition.append([True, [i[0]] + [j[1:] for j in
                              i[1:]]])
46                    else:
47                        self.precondition.append([False, [i[1][0]] + [j[1:] for
                              j in i[1][1:]]])
48            self.effect = []
49            for i in init_list[7]:
50                if type(i) is list:
51                    if i[0] != 'not':
```

```
52                      self.effect.append([True, [i[0]] + [j[1:] for j in i
                            [1:]]])
53                  else:
54                      self.effect.append([False, [i[1][0]] + [j[1:] for j in i
                            [1][1:]]])
55
56          # 寻找可能的动作实例，返回 [[ 动作名 , add_list , del_list ]...]
57          # 一种动作也有多种可能，这个 bug 找了一年
58          def do(self, objects, kb):
59              rst = []
60              mapping_list = []
61              self.crearte_mapping(objects, {}, 0, mapping_list)
62              for i in mapping_list:
63                  precondition = self.assign_precondition(i)
64                  flag = True
65                  for j in precondition:
66                      if (j[0] and j[1] not in kb) or (not j[0] and j[1] in kb):
67                          flag = False
68                          break
69                  if flag:
70                      rst.append(self.assign_effect(i))
71              return rst
72
73          # 启发式函数的辅助，返回 [[ 肯定的 precondition , add]...]
74          def relax_do(self, objects, kb):
75              rst = []
76              mapping_list = []
77              self.crearte_mapping(objects, {}, 0, mapping_list)
78              for i in mapping_list:
79                  precondition = self.assign_precondition(i)
80                  flag = True
81                  for j in precondition:
82                      if (j[0] and j[1] not in kb) or (not j[0] and j[1] in kb):
83                          flag = False
84                          break
85                  if flag:
86                      pre = []
```

```
87                  for j in range(len(precondition)):
88                      if precondition[j][0]:
89                          pre.append(precondition[j][1])
90                  rst.append([pre, self.assign_effect(i)[1]])
91          return rst
92
93      # 创建对象间的映射以实现实例化，返回映射的列表
94      def crearte_mapping(self, objects, curr, depth, rst):
95          if depth == len(self.parameters):
96              rst.append(copy.deepcopy(curr))
97              return
98          ind, val = list(self.parameters.items())[depth]
99          for i in objects[val]:
100             if i not in curr.values(): # 这里默认一个动作参数中不能出现相同对象
101                 curr[ind] = i
102                 self.crearte_mapping(objects, curr, depth+1, rst)
103                 curr.pop(ind)
104
105     # 实例化条件
106     def assign_precondition(self, mapping):
107         precondition = copy.deepcopy(self.precondition)
108         for i in range(len(precondition)):
109             for j in range(1, len(precondition[i][1])):
110                 precondition[i][1][j] = mapping[precondition[i][1][j]]
111         return precondition
112
113     # 实例化效果
114     def assign_effect(self, mapping):
115         effect = copy.deepcopy(self.effect)
116         add = []
117         delete = []
118         for i in range(len(effect)):
119             for j in range(1, len(effect[i][1])):
120                 effect[i][1][j] = mapping[effect[i][1][j]]
121             if effect[i][0]:
122                 add.append(effect[i][1])
123             else:
```

```
124                     delete.append(effect[i][1])
125             return [[self.name]+list(mapping.values()), add, delete]
126 
127         def printInfo(self):
128             print("\t{" + self.name + "}")
129             print("\tparameters:", self.parameters)
130             print("\tprecondition:", self.precondition)
131             print("\teffect:", self.effect)
132 
133 
134     class Strips:
135         def __init__(self, domain_filename, problem_filename):
136             domain = parser(domain_filename)
137             problem = parser(problem_filename)
138             self.types = ['AnyType']  # 没有类型时
139             self.actions = []
140             for i in domain:
141                 if type(i) is list:
142                     if i[0] == ':types':
143                         self.types = i[1:]
144                     if i[0] == ':action':
145                         self.actions.append(Action(i, self.types))
146             self.objects = {}
147             object = problem[3][1:]
148             tail = 0
149             if self.types != ['AnyType']:
150                 for ind, i in enumerate(object):
151                     if i == '-':
152                         self.objects[object[ind+1]] = object[tail:ind]
153                         tail = ind + 2
154             else:
155                 self.objects['AnyType'] = object
156             self.kb = problem[4][1:]
157             self.goal = []
158             goal = problem[5][1][1:]
159             for i in goal:
160                 if i[0] != 'not':
```

```
161                     self.goal.append([True, i])
162                 else:
163                     self.goal.append([False, i[1:]])
164             self.path = []
165 
166         # 使用 A* 搜索解决问题，返回拓展结点数和路径
167         def search(succ):
168             if len(succ) == 0:
169                 return 1, []
170             state, action, effect = min(succ, key=lambda x: x[2][3]+x[2][4])
171             if effect[3] > MAX_DEPTH: # 限制深度，在所给测例中其实没用
172                 return 1, []
173             succ.remove((state, action, effect))
174             state.step(effect)
175             if state.achive():
176                 return 1, state.path
177             for i in state.observe():
178                 if not Strips.inverse(effect, i[2]): # 阻止搜索在两点间反复踏步
179                     succ.append(i)
180                 # succ.append(i)
181 
182             rst = Strips.search(succ)
183             return rst[0]+1, rst[1]
184 
185         # 观察周围，获取动作信息，返回 [( 状态 , 动作 , [ 动作
                名 , add_list, del_list, g, h])...]
186         def observe(self):
187             tem = []
188             for i in self.actions:
189                 for can in i.do(self.objects, self.kb):
190                     can.append(len(self.path)) # g
191                     h = self.heuristic(can)
192                     can.append(self.heuristic(can)) # h
193                     tem.append((copy.deepcopy(self), i, can))
194             return tem
195 
196         # 应用 reachability_analysis 和 count_actions ，返回h
197         def heuristic(self, action):
```

```
198             curr_state = copy.deepcopy(self)
199             curr_state.step(action)
200             state_layer = [copy.deepcopy(curr_state.kb)]
201             action_layer = []
202             new_to_pre = {}
203             while(not curr_state.achive()):
204                 actions = []
205                 for i in curr_state.actions:
206                     for can in i.relax_do(curr_state.objects, curr_state.kb):
207                         can.append(i)
208                         actions.append(can) # (pre, add, action)
209                 action_layer.append(copy.deepcopy(actions))
210                 for i in actions:
211                     for j in i[1]:
212                         if j not in curr_state.kb:
213                             new_to_pre[str(tuple(j))] = i
214                             curr_state.kb.append(j)
215                 if curr_state.kb == state_layer[-1]: # kb 不再变化
216                     return 100
217                 state_layer.append(copy.deepcopy(curr_state.kb))
218             state_layer = [ [tuple(j) for j in i] for i in state_layer]
219 
220             return Strips.count_acts(new_to_pre, state_layer, action_layer,
                    curr_state.goal, state_layer[-1])
221 
222         # 利用递归来计算 h
223         def count_acts(new_to_pre, state_layer, action_layer, G, S):
224             if len(state_layer) == 1:
225                 return 0
226             state_layer.remove(S)
227             s1 = set(S)
228             s2 = set(state_layer[-1])
229             Gp = s1 & s2
230             Gn = s1 - s2
231             A = []
232             for i in Gn:
233                 action = new_to_pre[str(i)]
```

```
234                 if action[2] not in A:
235                     A.append(action[2])
236                     Gp = Gp | set([tuple(j) for j in action[0]])
237             return Strips.count_acts(new_to_pre, state_layer, action_layer, Gp,
                    state_layer[-1]) + len(A)
238
239         # 行动
240         def step(self, action):
241             self.path.append(action[0])
242             for i in action[1]: # add
243                 self.kb.append(i)
244             for i in action[2]: # del
245                 self.kb.remove(i)
246
247         # 判断是否达到目标
248         def achive(self):
249             for i in self.goal:
250                 if not i[0]: # 否定的目标
251                     for j in self.kb:
252                         if j == i[1]:
253                             return False
254                 else: # 肯定的目标
255                     for j in self.kb:
256                         find = False
257                         if j == i[1]:
258                             find = True
259                             break
260                     if not find:
261                         return False
262             return True
263
264         # 检查互逆动作以提升搜索效率
265         def inverse(effect1, effect2):
266             if effect1[0][0] != effect2[0][0]:
267                 return False
268             for i in effect1[1]:
269                 if i not in effect2[2]:
```

```
270                     return False
271             for i in effect2[1]:
272                 if i not in effect1[2]:
273                     return False
274             return True
275 
276         def printInfo(self):
277             print("{domain}")
278             print("types:", self.types)
279             print("actions:␣")
280             for curr in self.actions:
281                 curr.printInfo()
282             print("{problem}")
283             print("objects:", self.objects)
284             print("kb:", self.kb)
285             print("goal:", self.goal)
286             print()
287 
288     def test(num):
289         perfix = "pddl/test" + str(num) + "/test" + str(num)
290         strips = Strips(perfix + "_domain.pddl", perfix + "_problem.pddl")
291         begin = timer()
292         count, path = Strips.search(strips.observe())
293         end = timer()
294         length = len(path)
295         print("{test␣"+str(num)+"}")
296         print()
297         print("ANSWER:",end="␣")
298         for i in range(length):
299             print(path[i],end="->" if i != length-1 else "")
300             if (i+1) % 3 == 0 and i != length-1:
301                 print("\n\t",end="")
302         print()
303         print("Time␣Cost:␣"+str(end-begin)+"s")
304         print("Node␣Expended:", count)
305         print()
306 
```

```
307     def main():
308         for i in range(5):
309             test(i)
310
311     if __name__ == "__main__":
312         main()
```

- parser 是自己写的。

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

   - 在使用 A* 搜索时，不仅是选出 g+h 最小的动作，也会对动作进行筛选，不选和上次动作完全互逆的动作以防止“反复徘徊”。这是因为，可达性分析启发式函数不会删除知识，这导致搜素前期计算启发式函数时一些具有否定前提的动作永远不会被执行，这使得边界集合的启发式 h 值都很大，经常出现“反复徘徊”问题。而检查相邻动作是否完全互逆，可以解决这个问题。

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

   - 结果如下：针对每个测例，显示了行动序列，解决问题的时间，搜索时拓展的节点数。测例 0,1,3,4 的问题规模都比较小，测例 0，1 问题规模太小了，分别花了 2 步和 3 步直接搜到了结果，没有多走任何弯路，测例 3,4 也几乎是直接搜到了结果，都只多拓展了一个节点。测例 2 的目标路径有 10 步，拓展结点为 16 个，表现也很好，这里如果不使用“反复徘徊”检测，拓展结点数会稍微多一点，为 18 个。时间上，全部都是瞬间出结果，只有测例 2 稍微多一点，接近 1s。
   - 总得说所给测例都较小，但也能看出 A* 搜索和可达性分析启发式效果拔群。

```
{test 0}

ANSWER: ['move', 'npc', 'town', 'field']->['move', 'npc', 'field', 'castle']
Time Cost: 0.0016088000000000005s
Node Expended: 2

{test 1}

ANSWER: ['move', 'npc', 'town', 'tunnel']->['move', 'npc', 'tunnel', 'river']->['move', 'npc', 'river', 'castle']
Time Cost: 0.0065931000000000045s
Node Expended: 3
```

```
{test 2}

ANSWER: ['move', 'npc', 'town', 'field']->['attack', 'npc', 'ogre', 'field', 'river']->['move', 'npc', 'field', 'river']->
       ['attack', 'npc', 'dragon', 'river', 'cave']->['open', 'npc', 'box1', 'river']->['collect-fire', 'npc', 'box1', 'river', 'reddust']->
       ['move', 'npc', 'river', 'cave']->['open', 'npc', 'box2', 'cave']->['collect-earth', 'npc', 'box2', 'cave', 'browndust']->
       ['build-fireball', 'npc']
Time Cost: 0.7458731999999999s
Node Expended: 16
```

```
{test 3}

ANSWER: ['unstack', 'b', 'a', 'x']->['move', 'b', 'x', 'y']->['stack', 'a', 'b', 'x']
Time Cost: 0.034036999999999984s
Node Expended: 4

{test 4}

ANSWER: ['unstack', 'b', 'a', 't1', 't3']->['stack', 'a', 't1', 'b', 't3']
Time Cost: 0.3862719000000001s
Node Expended: 3
```

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)

   - 为了解决计算条件概率时复杂度 $O(2^{n-k})$ 问题 (n 为总变量数，k 为所求变量数)，使用 DP 想法，逐层计算条件概率，每层消除一个变量，将问题复杂度降至 $O(2^{O(w)})$（w 为贝叶斯网络的树宽，w«n）

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')

```
import copy
import random
import numpy as np
from timeit import default_timer as timer
num = 0

class VariableElimination:
    def printFactors(factorList):
        for factor in factorList:
            factor.printInf()

    def default_order(list, factorList):
        hold = []
        for i in list:
            hold.append(i)
            yield i
        print("default order:", hold)

    def random_order(list, factorList):
        length = len(list)
        hold = []
        for i in range(length):
            pick = random.choice(list)
            list.remove(pick)
            hold.append(pick)
            yield pick
        print("random order:", hold)

    def min_fill(list, factorList):
        pass

    def min_neighber(list, factorList):
```

```
33            length = len(list)
34            hold = []
35            for i in range(length):
36                var_list = [j.varList for j in factorList if len(j.varList)]
37                pick = random.choice(min(var_list, key=len))
38                hold.append(pick)
39                yield pick
40            print("min-neighber:", hold)
41
42        def min_weight(list, factorList):
43            length = len(list)
44            hold = []
45            for i in range(length):
46                var_map = dict(zip(list, [0]*(length-i)))
47                for node in factorList:
48                    for var in node.varList:
49                        if var in list:
50                            var_map[var] += 1
51                pick = min(var_map.items(), key=lambda x: x[1])[0]
52                hold.append(pick)
53                list.remove(pick)
54                yield pick
55            print("min-weight:", hold)
56
57        def inference(factorList, queryVariables,
58        orderedListOfHiddenVariables, evidenceList, queryVal, f):
59            for ev in evidenceList:
60                #Your code here
61                for pos, factor in enumerate(factorList):
62                    factorList[pos] = factor.restrict(ev, str(evidenceList[ev]))
                        # 注意这里需要转
                        成 str
63            print()
64            width = 0
65            for var in f(orderedListOfHiddenVariables, factorList):
66                #Your code here
67                temList= []
68                for pos, factor in enumerate(factorList):
```

```
69                  if var in factor.varList:
70                      temList.append(factor)
71              for factor in temList:
72                  factorList.remove(factor)
73              curr = temList[0]
74              for i in temList[1:]:
75                  curr = curr.multiply(i)
76              curr = curr.sumout(var)
77              factorList.append(curr)
78              if factorList != []:
79                  # 消除宽为最大超图大小
80                  max_width = max([len(i.varList) for i in factorList])
81                  width = width if max_width <= width else max_width
82          print("WIDTH:", width)
83          print("RESULT:")
84          res = factorList[0]
85          for factor in factorList[1:]:
86              res = res.multiply(factor)
87          total = sum(res.cpt.values())
88          res.cpt = {k: v/total for k, v in res.cpt.items()}
89          # res.printInf()
90          print(res.cpt[queryVal])
91
92  class Util:
93      def lfind(l, val):
94          for pos, i in enumerate(l):
95              if i==val:
96                  return pos
97          return -1
98
99  class Node:
100     def __init__(self, name, var_list):
101         self.name = name
102         self.varList = var_list
103         self.cpt = {}
104     def setCpt(self, cpt):
105         self.cpt = cpt
```

```
106         def printInf(self):
107             print("Name = " + self.name)
108             print("  vars " + str(self.varList))
109             for key in self.cpt:
110                 print("       key: " + str(key) + "  val : " + str(self.cpt[key]))
111             print()
112         def multiply(self, factor):
113             """function that multiplies with another factor"""
114             #Your code here
115             newList = list(set(self.varList+factor.varList))
116             length = len(newList)
117             new_cpt = {}
118             if self.varList == []:
119                 for key, val in factor.cpt.items():
120                     new_cpt[key] = val*self.cpt['']
121             elif factor.varList == []:
122                 for key, val in self.cpt.items():
123                     new_cpt[key] = val*factor.cpt['']
124             else:
125                 map1 = {}
126                 map2 = {}
127                 for pos, i in enumerate(self.varList):
128                     map1[pos] = Util.lfind(newList, i)
129                 for pos, i in enumerate(factor.varList):
130                     map2[pos] = Util.lfind(newList, i)
131                 convex = list(set(self.varList) & set(factor.varList))
132                 test = convex != []
133                 if test:
134                     repeated_var = convex[0]
135                     pos1 = Util.lfind(self.varList, repeated_var)
136                     pos2 = Util.lfind(factor.varList, repeated_var)
137 
138                 for key1, val1 in self.cpt.items():
139                     if test:
140                         repeated_ins = key1[pos1]
141                     for key2, val2 in factor.cpt.items():
142                         if test and repeated_ins == key2[pos2]:
```

```
143                             key = np.zeros(length, dtype=int)
144                             for ind, i in map1.items():
145                                 key[i] = key1[ind]
146                             for ind, i in map2.items():
147                                 key[i] = key2[ind]
148                             new_cpt["".join([str(num) for num in key])] = val1 *
                                    val2
149             new_node = Node("f" + str(newList), newList)
150             new_node.setCpt(new_cpt)
151             return new_node
152         def sumout(self, variable):
153             """function that sums out a variable given a factor"""
154             #Your code here
155             if variable not in self.varList:
156                 return self
157             pos = Util.lfind(self.varList, variable)
158             node_list = []
159             for i in set([j[pos] for j in self.cpt]):
160                 node_list.append(self.restrict(variable, i))
161             new_var_list = copy.deepcopy(node_list[0].varList)
162             new_cpt = copy.deepcopy(node_list[0].cpt)
163             for node in node_list[1:]:
164                 for key, p in node.cpt.items():
165                     new_cpt[key] += node.cpt[key]
166
167             new_node = Node("f" + str(new_var_list), new_var_list)
168             new_node.setCpt(new_cpt)
169             return new_node
170         def restrict(self, variable, value):
171             """function that restricts a variable to some value
172             in a given factor"""
173             #Your code here
174             if variable not in self.varList:
175                 return self
176             new_var_list = copy.deepcopy(self.varList) # 这里必须要用深复制
177             new_var_list.remove(variable)
178             length = len(new_var_list)
```

```
179             new_cpt = {}
180             pos = Util.lfind(self.varList, variable)
181             for key, p in self.cpt.items():
182                 if key[pos] == value:
183                     key = key[0:pos]+key[pos+1:]
184                     new_cpt[key] = p
185             new_node = Node("f" + str(new_var_list), new_var_list)
186             new_node.setCpt(new_cpt)
187             return new_node
188
189     def initBN():
190         # create nodes for Bayes Net
191         P = Node("PatientAge", ["P"])
192         C = Node("CTScanResult", ["C"])
193         MR = Node("MRIScanResult", ["MR"])
194         A = Node("Anticoagulants", ["A"])
195         S = Node("StrokeType", ["S", "C", "MR"])
196         MO = Node("Mortality", ["MO", "S", "A"])
197         D = Node("Disability", ["D", "S", "P"])
198
199         # Generate cpt for each node
200         P.setCpt({'0': 0.1, '1': 0.3, '2': 0.6})
201         C.setCpt({'0': 0.7, '1': 0.3})
202         MR.setCpt({'0': 0.7, '1': 0.3})
203         A.setCpt({'0': 0.5, '1': 0.5})
204         S.setCpt({'000': 0.8, '001': 0.5, '010': 0.5, '011': 0.0, \
205                   '100': 0.0, '101': 0.4, '110': 0.4, '111': 0.9, \
206                   '200': 0.2, '201': 0.1, '210': 0.1, '211': 0.1})
207         MO.setCpt({'000': 0.28, '010': 0.99, '020': 0.1, '001': 0.56, '011':
                0.58, '021': 0.05, \
208                    '100': 0.72, '110': 0.01, '120': 0.9, '101': 0.44, '111':
                       0.42, '121': 0.95})
209         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,
210                   '022': 0.10, \
211                   '100': 0.10, '110': 0.20, '120': 0.05, '101': 0.30, '111':
                      0.40, '121': 0.30, '102': 0.40, '112': 0.20,
```

```
212                 '122': 0.10, \
213                 '200': 0.10, '210': 0.10, '220': 0.05, '201': 0.10, '211':
                        0.10, '221': 0.30, '202': 0.30, '212': 0.60,
214                 '222': 0.80})
215         return [P, C, MR, A, S, MO, D]
216
217     def test(factor_list, f):
218         global num
219         print("ORDER␣"+str(num)+":")
220         num += 1
221         begin1 = timer()
222         VariableElimination.inference(copy.deepcopy(factor_list), ['A'], ['C', '
                MR', 'S', 'MO', 'D'], {'P': 1}, '1', f)
223         end1 = timer()
224         print("time1:", (end1 - begin1)*1000, "ms")
225         begin2 = timer()
226         VariableElimination.inference(copy.deepcopy(factor_list), ['D'], ['P', '
                C', 'MR', 'A', 'S', 'MO'], {}, '0', f)
227         end2 = timer()
228         print("time2:", (end2 - begin2)*1000, "ms")
229         print()
230         print()
231
232     def main():
233         factor_list = initBN()
234
235         # test
236         VariableElimination.inference(copy.deepcopy(factor_list), ['MO', 'C'], [
                'MR', 'A', 'S', 'D'], {'P': 1}, '10', VariableElimination.
                default_order)
237         VariableElimination.inference(copy.deepcopy(factor_list), ['D', 'C'], ['
                A', 'S', 'MO'], {'P': 2, 'MR': 1}, '11', VariableElimination.
                default_order)
238         VariableElimination.inference(copy.deepcopy(factor_list), ['S'], ['A', '
                MO', 'D'], {'P': 2, 'C': 1, 'MR': 0}, '1', VariableElimination.
                default_order)
239         VariableElimination.inference(copy.deepcopy(factor_list), ['A'], ['C', '
```

```
            MR', 'S', 'MO', 'D'], {'P': 1}, '1', VariableElimination.
            default_order)
240         VariableElimination.inference(copy.deepcopy(factor_list), ['D'], ['P', '
            C', 'MR', 'A', 'S', 'MO'], {}, '0', VariableElimination.default_order
            )
241         # test(factor_list, VariableElimination.default_order)
242         # test(factor_list, VariableElimination.random_order)
243         # test(factor_list, VariableElimination.random_order)
244         # test(factor_list, VariableElimination.random_order)
245         # test(factor_list, VariableElimination.min_neighber)
246         # test(factor_list, VariableElimination.min_weight)
247
248
249     if __name__ == "__main__":
250         main()
```

```
default order: ['MR', 'A', 'S', 'D']
WIDTH: 3
RESULT:
0.17587499999999995

default order: ['A', 'S', 'MO']
WIDTH: 3
RESULT:
0.057

default order: ['A', 'MO', 'D']
WIDTH: 2
RESULT:
0.39999999999999997

default order: ['C', 'MR', 'S', 'MO', 'D']
WIDTH: 3
RESULT:
0.5000000000000001

default order: ['P', 'C', 'MR', 'A', 'S', 'MO']
WIDTH: 3
RESULT:
0.38977
```

- 该代码基于实验九的框架。上图是在人为给定变量消除顺序下的测试，计算出了正确结果。

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

   - 见上方代码 VariableElimination 类内的 4 个 order 函数，同时需要把 main 内的原测试注释，将原注释取消注释进行 order 相关的测试。

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)

   - 测试结果如下：

```
ORDER 0:

default order: ['C', 'MR', 'S', 'MO', 'D']
WIDTH: 3
RESULT:
0.5
time1: 6.702700000000061 ms

default order: ['P', 'C', 'MR', 'A', 'S', 'MO']
WIDTH: 3
RESULT:
0.38977
time2: 5.2714999999999845 ms


ORDER 1:

random order: ['MR', 'D', 'S', 'MO', 'C']
WIDTH: 3
RESULT:
0.5
time1: 5.010199999999965 ms

random order: ['MR', 'A', 'S', 'C', 'MO', 'P']
WIDTH: 4
RESULT:
0.38977000000000006
time2: 12.750799999999952 ms
```

```
ORDER 2:

random order: ['D', 'MO', 'MR', 'S', 'C']
WIDTH: 3
RESULT:
0.5
time1: 3.326199999999946 ms

random order: ['S', 'P', 'MO', 'C', 'A', 'MR']
WIDTH: 6
RESULT:
0.38977
time2: 28.335099999999947 ms


ORDER 3:

random order: ['D', 'S', 'MO', 'C', 'MR']
WIDTH: 4
RESULT:
0.5
time1: 5.089499999999969 ms

random order: ['MO', 'A', 'P', 'S', 'MR', 'C']
WIDTH: 3
RESULT:
0.38977
time2: 3.269999999999995 ms
```

```
ORDER 4:

min-neighber: ['C', 'MR', 'A', 'S', 'MO']
WIDTH: 3
RESULT:
0.3249
time1: 2.4287999999998977 ms

min-neighber: ['P', 'C', 'MR', 'A', 'S', 'MO']
WIDTH: 3
RESULT:
0.38977
time2: 2.9660999999999715 ms


ORDER 5:

min-weight: ['MO', 'D', 'C', 'MR', 'S']
WIDTH: 3
RESULT:
0.5
time1: 1.79549999999995 ms

min-weight: ['MO', 'P', 'C', 'MR', 'A', 'S']
WIDTH: 3
RESULT:
0.38977
time2: 2.3180999999999896 ms
```

Test case	Elimination order	Elimination width	Total time(ms)
p4	[C,MR,S,MO,D]	3	6.7
p4	[MR,D,S,MO,C]	3	5.0
p4	[D,MO,MR,S,C]	3	3.2
p4	[D,S,MO,C,MR]	4	5.1
p4	[C,MR,A,S,MR]	3	2.4
p4	[MO,D,C,MR,S]	3	1.8
p5	[P,C,MR,A,S,MO]	3	5.3
p5	[MR,S,A,C,MO,P]	4	12.8
p5	[S,P,MO,C,A,MR]	6	28.3
p5	[MO,A,P,S,MR,C]	3	3.3
p5	[P,C,MR,A,S,MO]	3	3.0
p5	[MO,P,C,MR,A,S]	3	2.3

- 如图和表所示，对测例 4,5 使用了 6 种变量消除顺序，第一种是默认给出的顺序，第二至第四种顺序是随机顺序，第五种顺序是优先消除邻居数小的节点的变量，第六种顺序是优

先消除在图中节点所占权重小的变量。

- 总体上，消除宽度越大，运行时间越久。
- 两种算法（最少邻居，最小权重）都取得了不错的效果，最小权重算法获得了最小的时间。

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