



Inteligência Artificial e Sistemas de Decisão - 1º Semestre 2020/2021 (MECD MEAr MEEC)

[Dashboard](#) / [My courses](#) / [IASD2021](#) / [Mini-project #1](#) / [Code submission](#)

- [Description](#)
- [Submission](#)
- [Edit](#)
- Submission view

Submitted on Tuesday, 10 November 2020, 5:26 PM ([Download](#)) ([Evaluate](#))

Submitted by  Diogo Moura

 Rafael Benavente  Miguel Santos

Automatic evaluation[-]

Proposed grade: 0 / 48

Comments[-]

----- PUBLIC TESTS -----

*** Problem PUB1 ***

Execution time: 0.000006 secs

Output:

MD DOC 025 1.0

MD LAB 01 10 15 LAB 02 20 10 LAB 03 40 5

MD PatientID:001 currWaitTime:0 label:01

Solved with cost -1

Validation: ERR: Number of Medical Doctors differs from the problem

GRADE: 0 point(s)

*** Problem PUB2 ***

Execution time: 0.000006 secs

Output:

MD DOC 025 1.0

MD LAB 01 10 15 LAB 02 20 10 LAB 03 40 5

MD PatientID:001 currWaitTime:5 label:01

Solved with cost -1

Validation: ALG: Problem infeasable, ALG gives something else

GRADE: 0 point(s)

*** Problem PUB3 ***

Execution time: 0.000005 secs

Output:

MD DOC 025 1.0 DOC 005 1.0

MD LAB 01 5 10 LAB 02 20 5 LAB 03 35 5

MD PatientID:001 currWaitTime:5 label:01PatientID:002 currWaitTime:0 label:01

Solved with cost -1

Validation: ERR: Number of Medical Doctors differs from the problem

GRADE: 0 point(s)

*** Problem PUB4 ***

Execution time: 0.000006 secs

Output:

MD DOC 0001 1.0 DOC 0002 0.25

MD LAB 01 10 15 LAB 02 20 10 LAB 03 40 5

MD PatientID:001 currWaitTime:0 label:01PatientID:002 currWaitTime:15 label:03

Solved with cost -1

Validation: ERR: Number of Medical Doctors differs from the problem

GRADE: 0 point(s)

*** Problem PUB5 ***

Execution time: 0.000007 secs

Output:

MD DOC 0001 1.0 DOC 0002 0.25 DOC 0003 0.25

MD LAB 01 10 15 LAB 02 20 10 LAB 03 40 5

MD PatientID:001 currWaitTime:0 label:01PatientID:002 currWaitTime:15 label:03PatientID:003 currWaitTime:0 label:02

Solved with cost -1

Validation: ERR: Depths are inconsistent
GRADE: 0 point(s)

*** Problem PUB6 ***

Execution time: 0.000007 secs

Output:

MD DOC 0001 1.0 DOC 0003 0.5 DOC 0002 0.5 DOC 0004 1.0
MD LAB 01 10 15 LAB 02 20 10 LAB 03 40 5
MD PatientID:001 currWaitTime:0 label:01PatientID:002 currWaitTime:15 label:03PatientID:003 currWaitTime:0 label:02PatientID:004 currWaitTime:5 label:01
Solved with cost -1
Validation: ERR: Number of Medical Doctors differs from the problem
GRADE: 0 point(s)

*** Problem PUB7 ***

Execution time: 0.000009 secs

Output:

MD DOC 0001 1.0 DOC 0003 0.5 DOC 0002 0.5
MD LAB 01 10 15 LAB 02 20 10 LAB 03 40 5
MD PatientID:001 currWaitTime:0 label:01PatientID:002 currWaitTime:5 label:03PatientID:003 currWaitTime:5 label:02
Solved with cost -1
Validation: ERR: Depths are inconsistent
GRADE: 0 point(s)

*** Problem PUB8 ***

Execution time: 0.000008 secs

Output:

MD DOC 0001 1.0 DOC 0004 1.0 DOC 0002 1.0 DOC 0010 1.0
MD LAB 01 15 15 LAB 02 25 10 LAB 03 35 5
MD PatientID:001 currWaitTime:0 label:03PatientID:002 currWaitTime:5 label:01PatientID:003 currWaitTime:5 label:02PatientID:004 currWaitTime:5 label:03
Solved with cost -1
Validation: ERR: Number of Medical Doctors differs from the problem
GRADE: 0 point(s)

>> PUBLIC TESTS GRADE: 0 <<<

----- PRIVATE TESTS -----

*** Problem PVT1 ***

Execution time: 0.000011 secs

Output:

MD DOC Joao 0.5 DOC Daniel 1.0 DOC Pedro 0.5 DOC Ricardo 1.0
MD LAB urg 10 15 LAB mid 20 10
MD PatientID:Carlos currWaitTime:0 label:urgPatientID:Jose currWaitTime:15 label:mid
Solved with cost -1
Validation: ERR: Number of Medical Doctors differs from the problem
GRADE: 0 point(s)

*** Problem PVT2 ***

Execution time: 0.000014 secs

Output:

MD DOC Joao 0.5 DOC Daniel 1.0 DOC Pedro 0.5
MD LAB urg 10 15 LAB mid 20 10
MD PatientID:Carlos currWaitTime:0 label:urgPatientID:Jose currWaitTime:15 label:midPatientID:Carlos currWaitTime:0 label:urg
Solved with cost -1
Validation: ERR: Depths are inconsistent
GRADE: 0 point(s)

*** Problem PVT3 ***

Execution time: 0.000015 secs

Output:

MD DOC 0001 1.0 DOC 0002 1.0 DOC 0003 1.0 DOC 0004 1.0
MD LAB 01 10 15 LAB 02 20 10 LAB 03 40 5
MD PatientID:001 currWaitTime:0 label:01PatientID:002 currWaitTime:15 label:03PatientID:003 currWaitTime:0 label:02PatientID:004 currWaitTime:5 label:01
Solved with cost -1
Validation: ERR: Number of Medical Doctors differs from the problem
GRADE: 0 point(s)

*** Problem PVT4 ***

Execution time: 0.000015 secs

Output:

MD DOC 0001 0.5 DOC 0002 0.5 DOC 0003 0.5 DOC 0004 0.5
MD LAB 01 10 15 LAB 02 20 10 LAB 03 40 5
MD PatientID:001 currWaitTime:0 label:01PatientID:002 currWaitTime:15 label:03PatientID:003 currWaitTime:0 label:02PatientID:004 currWaitTime:5 label:01

```
-----
Solved with cost -1
Validation: ERR: Numnal of Medical Doctors differs from the problem
GRADE: 0 point(s)
*** Problem PVT5 ***
Execution time: 0.000007 secs
Output:
MD DOC 0001 1.0
MD LAB 01 10 15 LAB 02 20 10 LAB 03 40 5
MD PatientID:001 currWaitTime:0 label:01
Solved with cost -1
Validation: ALG: Problem infeasable, ALG gives something else
GRADE: 0 point(s)
*** Problem PVT6 ***
Execution time: 0.000018 secs
Output:
MD DOC 0001 1.0 DOC 0002 1.0 DOC 0003 0.25
MD LAB 01 30 10
MD PatientID:001 currWaitTime:0 label:01PatientID:002 currWaitTime:5 label:01PatientID:003 currWaitTime:10 label:01
Solved with cost -1
Validation: ERR: Depths are inconsistent
GRADE: 0 point(s)
*** Problem PVT7 ***
Execution time: 0.000015 secs
Output:
MD DOC 0001 1.0 DOC 0002 1.0 DOC 0003 0.5
MD LAB 01 5 5 LAB 02 10 5
MD PatientID:001 currWaitTime:5 label:02PatientID:002 currWaitTime:5 label:01PatientID:003 currWaitTime:10 label:02
Solved with cost -1
Validation: ERR: Depths are inconsistent
GRADE: 0 point(s)
*** Problem PVT8 ***
Execution time: 0.000015 secs
Output:
MD DOC 025 1.0 DOC 026 1.0 DOC 027 1.0
MD LAB 01 10 15 LAB 02 20 10 LAB 03 40 5
MD PatientID:001 currWaitTime:0 label:01PatientID:002 currWaitTime:0 label:01PatientID:003 currWaitTime:0 label:01
Solved with cost -1
Validation: ERR: Depths are inconsistent
GRADE: 0 point(s)
>> PRIVATE TESTS GRADE: 0 <<<
```

----- THE END -----

solution.py

```

1 from search import Problem
2 from search import uniform_cost_search
3 from search import astar_search
4 from itertools import permutations, combinations
5 import time
6 import math
7 import sys
8
9 class Doctor():
10     def __init__(self, _id, efficiency):
11         self._id = _id
12         self.efficiency = efficiency
13
14
15
16 class PatientLabel():
17     def __init__(self, max_wait_time, consult_time):
18         self.maxWaitTime = max_wait_time
19         self.consult_time = consult_time
20
21
22 class Patient():
23     def __init__(self, _id, curr_wait_time, label, remain_consult_time, max_wait_time):
24         self.labelID = label
25         self._id = _id
26         self.currWaitTime = curr_wait_time
27         self.remainConsultTime = remain_consult_time
28         self.maxWaitTime = max_wait_time
29
30     def toString(self):
31         return ("ID:" + self._id + " currWaitTime:" + str(self.currWaitTime) +
32 " label:" + self.labelID)
33
34     def copy(self):
35         return Patient(self._id, self.currWaitTime, self.labelID, self.remainConsultTime, self.maxWaitTime)
36
37     def __eq__(self, other):
38         if (self.currWaitTime == other.currWaitTime and self.remainConsultTime == other.remainConsultTime):
39             return True
40         return False
41
42
43 class State():
44     def __init__(self, patient_list, pathCost, docAssignment):
45         self.patient_list = patient_list
46         self.path_cost = pathCost
47         self.doctor_assignment = docAssignment
48         #[(1,15,30,15),(2,12,40,20),...] (#patient_id, curr_wait_time, max_wait_time, remain_consult_time)
49         #[(1,15,3),(2,12,2),...] (#patient_id, curr_wait_time, #label, remain_consult_time)
50         #state.numb_doctors
51         #self.labels = labels #{'labelid':(max_wait_time)}
52
53
54     def toString(self):
55         result = "--State--"
56         for patient in self.patient_list:
57             result += ("\n" + patient.toString())
58         return result
59
60     def copy(self):
61         newStateList = []
62         for patient in self.patient_list:
63             new_patient = patient.copy()
64             newStateList.append(new_patient)
65         doctor_list = self.doctor_assignment.copy()
66         return State(newStateList, self.path_cost, doctor_list)
67
68     def __lt__(self, state):
69         return self.path_cost < state.path_cost
70
71     def __eq__(self, other):
72         if (other.patient_list == None):
73             return True
74         for i in range(len(self.patient_list)):
75             if self.patient_list[i] != other.patient_list[i]:
76                 return False
77         return True
78
79     def __hash__(self):
80         # We use the hash value of the state
81         # stored in the node instead of the node
82         # object itself to quickly search a node
83         # with the same state in a Hash Table
84         return hash(self.path_cost)
85
86
87
88
89 def goal_test(self, state):
90     Returns True if state s is a goal state, and False otherwise
91
92     for patient in state.patient_list :
93         if (patient.currWaitTime > patient.maxWaitTime) or (patient.remainConsultTime != 0):
94             return False
95     return True
96
97
98
99
100 def __lt__(self, state): #put in front possible nodes or with smaller path costs
101     #if inserted Node is impossible
102     if self.goal_test(self) == False:

```

```

102         if self.goal_test(self)==False:
103             return False
104         #if other Node is impossible
105
106     elif self.goal_test(state)==False:
107         return True
108     #if both nodes are possible
109     else:
110         #Check which has greater cost
111         state1Cost=0
112         for patient in self.patient_list:
113             state1Cost+=patient.currWaitTime*patient.currWaitTime
114         state2Cost=0
115         for patient in state.patient_list:
116             state2Cost+=patient.currWaitTime*patient.currWaitTime
117         return state1Cost<state2Cost#return True when new state has smaller cost than others
118     ...
119
120 class PDMAProblem(Problem):
121     def __init__(self):
122         super().__init__(None)
123         self.labels=dict()
124         self.doctor_dict=dict()
125         self.initial=None
126         self.nodes_expanded=0
127         self.numb_docs=0
128         self.solution=None
129
130     def actions(self,state):
131         ...
132         Returns a list (or a generator) of operators applicable to state s
133         ...
134         doctor_ids=self.doctor_dict.keys()
135         ...
136         if(state.patient_list==None):
137             return list()
138         patients_on_limit=[]
139         for patient in state.patient_list:
140             if patient.currWaitTime == self.labels[patient.labelID].maxWaitTime:#pruning
141                 patients_on_limit.append(patient._id)
142         #with_ids
143         #print("ON LIMIT: ",patients_on_limit)
144         if len(patients_on_limit)!=0:
145             if len(patients_on_limit)>self.numb_docs:
146                 return list()
147             if len(patients_on_limit)==self.numb_docs:
148                 patient_ids=patients_on_limit
149                 _min=min(len(doctor_ids),len(patient_ids))
150                 possibleActions = [dict(zip(x,doctor_ids)) for x in permutations(patient_ids,_min)]
151             else:
152                 patient_ids=[patient._id for patient in state.patient_list if patient.remainConsultTime>0]
153                 _min=min(len(doctor_ids),len(patient_ids))
154                 permuts=permutations(patient_ids,_min)
155                 #print("Permut")
156                 #for p in permuts:
157                     # print(p)
158                 _permuts=[]
159                 for permutation in permuts:
160                     for on_limit in patients_on_limit:
161                         if on_limit not in permutation:
162                             continue
163                         _permuts.append(permutation)
164                 #print("_Permut")
165                 #for p in _permuts:
166                     # print(p)
167                 possibleActions = [dict(zip(x,doctor_ids)) for x in _permuts]
168                 #print(possibleActions)
169             ...
170         if state.patient_list==None:
171             return []
172         #else:
173         patient_ids=[patient._id for patient in state.patient_list if patient.remainConsultTime>0]
174         _min=min(len(doctor_ids),len(patient_ids))
175         permuts=permutations(patient_ids,_min)
176         possibleActions = [dict(zip(x,doctor_ids)) for x in permuts]
177         #print(state.toString())
178         #print(possibleActions)
179         return possibleActions
180
181     def result(self,state,action):
182         ...
183         Returns the state resulting from applying action a to state s
184         ...
185         newState=state.copy()
186         #newState=State(state.patient_list.copy())
187         #print(newState.toString())
188         for patient in newState.patient_list:
189             if patient.remainConsultTime != 0 :
190                 try:
191                     doc_id=action[patient._id]
192                     patient.remainConsultTime=max(0,patient.remainConsultTime-self.doctor_dict[doc_id].efficiency*5)
193                 except KeyError:
194                     patient.currWaitTime+=5
195             if patient.currWaitTime > self.labels[patient.labelID].maxWaitTime:#pruning
196                 newState.patient_list=None
197                 newState.path_cost=float('inf')
198                 self.nodes_expanded+=1
199             return newState
200
201         newState.doctor_assignment.append(action)
202
203         newState.path_cost=self.path_cost(state.path_cost,state,action,newState)
204         self.nodes_expanded+=1
205         return newState
206

```

```

207 def goal_test(self, state):
208     """
209     Returns True if state s is a goal state, and False otherwise
210     """
211     if state.patient_list==None:
212         return False
213     for patient in state.patient_list :
214         if (patient.currWaitTime > self.labels[patient.labelID].maxWaitTime) or (patient.remainConsultTime != 0):
215             return False
216     return True
217
218 def path_cost(self, cost, state1, action, state2):
219     """
220     Returns the path cost of state s2, reached from state s1 by
221     applying action a, knowing that the path cost of s1 is c.
222     We consider the following cost associated to all the patients in the waiting room:
223      $C(P) = \sum(p \text{ in } P) (p_{cw})^2$  where  $p_{cw}$  is the patient waiting time
224     """
225     state1Cost=0
226     for patient in state1.patient_list:
227         state1Cost+=patient.currWaitTime*patient.currWaitTime
228     state2Cost=0
229     for patient in state2.patient_list:
230         state2Cost+=patient.currWaitTime*patient.currWaitTime
231     return state2Cost
232
233 def load(self, file):
234     """
235     Loads a problem from a (opened) file object f (see below for format specification)
236     """
237     patient_list=list()
238     doctor_assignments=list()
239
240     for line in file:
241         line = line.rstrip()
242         if not line:
243             continue
244         info=line.split()
245         if (info[0]=='MD'):
246             #dict with keys as doc_id and doctors as values
247             self.doctor_dict[info[1]]=Doctor(info[1],float(info[2]))
248         elif(info[0]=='PL'):
249             self.labels[info[1]]=PatientLabel(int(info[2]),int(info[3]))
250         elif(info[0]=='P'):
251             patient_list.append(Patient(info[1],int(info[2]),
252                                     info[3],int(self.labels[info[3]].consult_time),
253                                     self.labels[info[3]].maxWaitTime))
254     self.numb_docs=len(self.doctor_dict.keys())
255     self.initial=State(patient_list,0,doctor_assignments)
256
257 def save(self, f):
258     i=0
259     f.write('MD ')
260     for doctor in self.doctor_dict:
261         f.write('DOC ')
262         d=self.doctor_dict[doctor]._id+' '+str(self.doctor_dict[doctor].efficiency)+' '
263         f.write(d)
264     f.write('\n')
265     f.write('MD')
266     for label in self.labels.keys():
267         f.write(' LAB ')
268         f.write(label)
269         f.write(' ')
270         f.write(str(self.labels[label].maxWaitTime))
271         f.write(' ')
272         f.write(str(self.labels[label].consult_time))
273     f.write('\n')
274     f.write('MD ')
275     for patient in self.initial.patient_list:
276         f.write('Patient')
277         f.write(patient.toString())
278         print(patient.toString())
279         i+=1
280         if i==self.numb_docs:
281             break
282     f.write('\n')
283
284     for patient in problem.initial.patient_list:
285         print('(', patient._id, " currW:", patient.currWaitTime, " maxW:", patient.maxWaitTime,
286             " remainC:", patient.remainConsultTime, ')')
287     print('\nDoctors:')
288     for doctor in problem.doctor_dict:
289         print('(', problem.doctor_dict[doctor]._id, problem.doctor_dict[doctor].efficiency, ')')
290     print('\nLabels:', problem.labels)
291
292 def search(self, **kwargs):
293     return True
294
295 def heuristic(self, node):
296     return 0

```

◀ [Announcements](#)

Jump to...

[Report](#) ▶

[VPL](#)

You are logged in as Diogo Moura (Log out)
IASD2021
Data retention summary
Get the mobile app