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

Submitted on Wednesday, 11 November 2020, 11:05 PM (Download) (Evaluate)

Submitted by Rafael Benavente

Diogo Moura Miguel Santos

Automatic evaluation[-]

Proposed grade: 0 / 48

Comments[-]

----- PUBLIC TESTS -----
*** Problem PUB1 ***
Execution time: 0.000005 secs
Output:
MD newline MD 025 1 newline newline PL 01 10 15 newline PL 02 20 10 newline PL 03 40 5 newline newline P 001 0 01 newline P 002 15 03 newline P 003 0 02 newline
Something went wrong:
Traceback (most recent call last):
File "evaluate.py", line 499, in worker
ret = run(pkey)
File "evaluate.py", line 478, in run
(grade, message) = val.check_solution(cost, succ)
File "evaluate.py", line 400, in check_solution
patients_consult_time_update[patient] += self.time_interval * self.medical_doctors[md]
KeyError: 'MD'
Unknown error
GRADE: 0 point(s)
*** Problem PUB2 ***
Execution time: 0.000005 secs
Output:
MD newline MD 025 1 newline newline PL 01 10 15 newline PL 02 20 10 newline PL 03 40 5 newline newline P 001 5 01 newline P 002 5 01 newline P 003 0 03 newline
Solved with cost -1
Validation: ALG: Problem infeasable, ALG gives something else
GRADE: 0 point(s)
*** Problem PUB3 ***
Execution time: 0.000006 secs
Output:
MD newline MD 025 1 newline MD 005 1 newline newline PL 01 5 10 newline PL 02 20 5 newline PL 03 35 5 newline newline P 001 5 01 newline P 002 0 01 newline P 003 25 03 newline P 020 5 02 newline
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 newline MD 0001 1 newline MD 0002 0.25 newline newline PL 01 10 15 newline PL 02 20 10 newline PL 03 40 5 newline newline P 001 0 01 newline P 002 15 03 newline P 003 0 02 newline P 004 5 01 newline P 005 10 03 newline P 006 15 03 newline
Solved with cost -1
Validation: ERR: Number of Medical Doctors differs from the problem
GRADE: 0 point(s)
*** Problem PUB5 ***
Execution time: 0.000006 secs
Output:

MD newline MD 0001 1 newline MD 0002 0.25 newline MD 0003 0.25 newline newline PL 01 10 15 newline PL 02 20 10 newline PL 03 40 5 newline newline P 001 0 01 newline P 002 15 03 newline P 003 0 02 newline P 004 5 01 newline P 005 10 03 newline P 006 15 03 newline P 009 15 03 newline
Solved with cost -1
Validation: ERR: Number of Medical Doctors differs from the problem
GRADE: 0 point(s)
*** Problem PUB6 ***
Execution time: 0.000008 secs
Output:
MD newline MD 0001 1 newline MD 0003 0.5 newline MD 0002 0.5 newline MD 0004 1 newline newline PL 01 10 15 newline PL 02 20 10 newline PL 03 40 5 newline newline P 001 0 01 newline P 002 15 03 newline P 003 0 02 newline P 004 5 01 newline P 005 15 03 newline P 006 0 02 newline P 007 5 01 newline P 008 10 03 newline
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 newline MD 0001 1 newline MD 0003 0.5 newline MD 0002 0.5 newline newline PL 01 10 15 newline PL 02 20 10 newline PL 03 40 5 newline newline P 001 0 01 newline P 002 5 03 newline P 003 5 02 newline P 004 5 01 newline P 005 0 03 newline P 006 0 02 newline
Solved with cost -1
Validation: ERR: Number of Medical Doctors differs from the problem
GRADE: 0 point(s)
*** Problem PUB8 ***
Execution time: 0.000011 secs
Output:
MD newline MD 0001 1 newline MD 0004 1 newline MD 0002 1 newline MD 0010 1 newline newline PL 01 15 15 newline PL 02 25 10 newline PL 03 35 5 newline newline P 001 0 03 newline P 002 5 01 newline P 003 5 02 newline P 004 5 03 newline P 005 10 03 newline P 006 5 02 newline P 007 5 01 newline
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 MD Joao 0.5 newline MD Daniel 1 newline MD Pedro 0.5 newline MD Ricardo 1 newline newline PL urg 10 15 newline PL mid 20 10 newline newline P Carlos 0 urg newline P Jose 15 mid newline
Solved with cost -1
Validation: ERR: Number of Medical Doctors differs from the problem
GRADE: 0 point(s)
*** Problem PVT2 ***
Execution time: 0.000013 secs
Output:
MD newline MD Joao 0.5 newline MD Daniel 1 newline MD Pedro 0.5 newline PL urg 10 15 newline PL mid 20 10 newline P Jose 15 mid newline P Carlos 0 urg newline P 001 15 mid newline P 002 0 mid newline P 1 0 urg newline
Solved with cost -1
Validation: ERR: Number of Medical Doctors differs from the problem
GRADE: 0 point(s)
*** Problem PVT3 ***
Execution time: 0.000014 secs
Output:
MD newline MD 0001 1 newline MD 0002 1 newline MD 0003 1 newline MD 0004 1 newline newline PL 01 10 15 newline PL 02 20 10 newline PL 03 40 5 newline newline P 001 0 01 newline P 002 15 03 newline P 003 0 02 newline P 004 5 01 newline P 005 10 03 newline P 006 15 03 newline P 009 15 03 newline P 010 5 03 newline P 011 0 02 newline P 020 0 01 newline
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 newline MD 0001 0.5 newline MD 0002 0.5 newline MD 0003 0.5 newline MD 0004 0.5 newline newline PL 01 10 15 newline PL 02 20 10 newline PL 03 40 5 newline newline P 001 0 01 newline P 002 15 03 newline P 003 0 02 newline P 004 5 01 newline P 005 10 03 newline P 006 15 03 newline P 009 15 03 newline
Solved with cost -1
Validation: ERR: Number of Medical Doctors differs from the problem
GRADE: 0 point(s)

GRADE: 0 point(s)
*** Problem PVT5 ***
Execution time: 0.000014 secs
Output:
MD newline MD 0001 1 newline newline PL 01 10 15 newline PL 02 20 10 newline PL 03 40 5 newline newline P 001 0 01 newline P 002 0 03 newline P 003 0 02 newline P 004 0 01 newline P 005 0 03 newline P 006 0 03 newline P 009 0 03 newline P 010 0 03 newline P 011 0 02 newline P 020 0 01 newline P 101 0 01 newline P 102 0 03 newline P 103 0 02 newline P 104 0 01 newline P 105 0 03 newline P 106 0 03 newline P 109 0 03 newline P 110 0 03 newline P 111 0 02 newline P 120 0 01 newline P 201 0 01 newline P 202 0 03 newline P 203 0 02 newline P 204 0 01 newline P 205 0 03 newline P 206 0 03 newline P 209 0 03 newline P 210 0 03 newline P 211 0 02 newline P 220 0 01 newline P 301 0 01 newline P 302 0 03 newline P 303 0 02 newline P 304 0 01 newline P 305 0 03 newline P 306 0 03 newline P 309 0 03 newline P 310 0 03 newline P 311 0 02 newline P 320 0 01 newline P 401 0 01 newline P 402 0 03 newline P 403 0 02 newline P 404 0 01 newline P 405 0 03 newline
Solved with cost -1
Validation: ALG: Problem infeasable, ALG gives something else
GRADE: 0 point(s)
*** Problem PVT6 ***
Execution time: 0.000014 secs
Output:
MD newline MD 0001 1 newline MD 0002 1 newline MD 0003 0.25 newline newline PL 01 30 10 newline newline P 001 0 01 newline P 002 5 01 newline P 003 10 01 newline P 004 5 01 newline P 005 0 01 newline P 006 5 01 newline P 009 5 01 newline
Solved with cost -1
Validation: ERR: Number of Medical Doctors differs from the problem
GRADE: 0 point(s)
*** Problem PVT7 ***
Execution time: 0.000015 secs
Output:
MD newline MD 0001 1 newline MD 0002 1 newline MD 0003 0.5 newline newline PL 01 5 5 newline PL 02 10 5 newline newline P 001 5 02 newline P 002 5 01 newline P 003 10 02 newline P 004 5 01 newline P 005 5 02 newline
Solved with cost -1
Validation: ERR: Number of Medical Doctors differs from the problem
GRADE: 0 point(s)
*** Problem PVT8 ***
Execution time: 0.000015 secs
Output:
MD newline MD 025 1 newline MD 026 1 newline MD 027 1 newline newline PL 01 10 15 newline PL 02 20 10 newline PL 03 40 5 newline newline P 001 0 01 newline P 002 0 01 newline P 003 0 01 newline
Solved with cost -1
Validation: ERR: Number of Medical Doctors differs from the problem
GRADE: 0 point(s)

```
>> PRIVATE TESTS GRADE: 0 <<<
```

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

solution.py

```python
from search import Problem
from search import uniform_cost_search
from search import astar_search
from itertools import permutations,combinations
import time
import math
import sys

class Doctor():
    def __init__(self,_id,efficiency):
        self._id=_id
        self.efficiency=efficiency



class PatientLabel():
    def __init__(self,max_wait_time,consult_time):
        self.maxWaitTime=max_wait_time
        self.consult_time=consult_time


class Patient():
    def __init__(self,_id,curr_wait_time,label,remain_consult_time,max_wait_time):
        self.labelID = label
        self._id=_id
        self.currWaitTime=curr_wait_time
        self.remainConsultTime=remain_consult_time
        self.maxWaitTime=max_wait_time

    def toString(self):
        return ("ID:"+self._id+" currWaitTime:"+str(self.currWaitTime)+
    " label:"+self.labelID)

    def copy(self):
        return Patient(self._id,self.currWaitTime,self.labelID,self.remainConsultTime,self.maxWaitTime)

    def __eq__(self,other):
        if(self.currWaitTime==other.currWaitTime and self.remainConsultTime==other.remainConsultTime):
            return True
        return False


class State():
    def __init__(self,patient_list,pathCost,docAssignment):
        self.patient_list=patient_list
        self.path_cost=pathCost
        self.doctor_assignment=docAssignment
        #[(1,15,30,15),(2,12,40,20),...] (#patient_id,curr_wait_time,max_wait_time,remain_consult_time)
        #[(1,15,3),(2,12,2),...] (#patient_id,curr_wait_time,#label,remain_consult_time)
        #state.numb_doctors
        #self.labels=labels #{'labelid':(max_wait_time)}


    def toString(self):
        result="--State--"
        for patient in self.patient_list:
            result+=("\n"+patient.toString())
        return result

    def copy(self):
        newStateList=[]
        for patient in self.patient_list:
            new_patient=patient.copy()
            newStateList.append(new_patient)
        doctor_list=self.doctor_assignment.copy()
        return State(newStateList,self.path_cost,doctor_list)

    def __lt__(self,state):
        return self.path_cost<state.path_cost

    def __eq__(self, other):
        if(other.patient_list == None):
            return True
        for i in range(len(self.patient_list)):
            if self.patient_list[i]!=other.patient_list[i]:
                return False
        return True

    def __hash__(self):
        # We use the hash value of the state
        # stored in the node instead of the node
        # object itself to quickly search a node
        # with the same state in a Hash Table
        return hash(self.path_cost)



    '''
    def goal_test(self,state):

        Returns True if state s is a goal state, and False otherwise

        for patient in state.patient_list :
            if (patient.currWaitTime > patient.maxWaitTime) or (patient.remainConsultTime != 0):
                return False
        return True
    '''

    '''
    def __lt__(self,state):#put in front possible nodes or with smaller path costs
        #if inserted Node is impossible
        if self.goal_test(self)==False:
```

```
        if self.goal_test(self)==False:
            return False
        #if other Node is impossible

        elif self.goal_test(state)==False:
            return True
        #if both nodes are possible
        else:
            #Check which has greater cost
            state1Cost=0
            for patient in self.patient_list:
                state1Cost+=patient.currWaitTime*patient.currWaitTime
            state2Cost=0
            for patient in state.patient_list:
                state2Cost+=patient.currWaitTime*patient.currWaitTime
            return state1Cost<state2Cost#return True when new state has smaller cost than others
        '''


class PDMAProblem(Problem):
    def __init__(self):
        super().__init__(None)
        self.labels=dict()
        self.doctor_dict=dict()
        self.initial=None
        self.nodes_expanded=0
        self.numb_docs=0
        self.solution=None

    def actions(self,state):
        '''
        Returns a list (or a generator) of operators applicable to state s
        '''
        doctor_ids=self.doctor_dict.keys()
        '''
        if(state.patient_list==None):
            return list()
        patients_on_limit=[]
        for patient in state.patient_list:
            if patient.currWaitTime == self.labels[patient.labelID].maxWaitTime:#pruning
                patients_on_limit.append(patient._id)
        #with_ids
        #print("ON LIMIT: ",patients_on_limit)
        if len(patients_on_limit)!=0:
            if len(patients_on_limit)>self.numb_docs:
                return list()
            if len(patients_on_limit)==self.numb_docs:
                patient_ids=patients_on_limit
                _min=min(len(doctor_ids),len(patient_ids))
                possibleActions = [dict(zip(x,doctor_ids)) for x in permutations(patient_ids,_min)]
            else:
                patient_ids=[patient._id for patient in state.patient_list if patient.remainConsultTime>0]
                _min=min(len(doctor_ids),len(patient_ids))
                permuts=permutations(patient_ids,_min)
                #print("Permuts")
                #for p in permuts:
                #    print(p)
                _permuts=[]
                for permutation in permuts:
                    for on_limit in patients_on_limit:
                        if on_limit not in permutation:
                            continue
                        _permuts.append(permutation)
                #print("_Permuts")
                #for p in _permuts:
                #    print(p)
                possibleActions = [dict(zip(x,doctor_ids)) for x in _permuts]
                #print(possibleActions)
        '''
        if state.patient_list==None:
            return []
        #else:
        patient_ids=[patient._id for patient in state.patient_list if patient.remainConsultTime>0]
        _min=min(len(doctor_ids),len(patient_ids))
        permuts=permutations(patient_ids,_min)
        possibleActions = [dict(zip(x,doctor_ids)) for x in permuts]
        #print(state.toString())
        #print(possibleActions)
        return possibleActions

    def result(self,state,action):
        '''
        Returns the state resulting from applying action a to state s
        '''
        newState=state.copy()
        #newState=State(state.patient_list.copy())
        #print(newState.toString())
        for patient in newState.patient_list:
            if patient.remainConsultTime != 0 :
                try:
                    doc_id=action[patient._id]
                    patient.remainConsultTime=max(0,patient.remainConsultTime-self.doctor_dict[doc_id].efficiency*5)
                except KeyError:
                    patient.currWaitTime+=5
            if patient.currWaitTime > self.labels[patient.labelID].maxWaitTime:#pruning
                newState.patient_list=None
                newState.path_cost=float('inf')
                self.nodes_expanded+=1
                return newState

        newState.doctor_assignment.append(action)

        newState.path_cost=self.path_cost(state.path_cost,state,action,newState)
        self.nodes_expanded+=1
        return newState
```

```
    def goal_test(self,state):
        '''
        Returns True if state s is a goal state, and False otherwise
        '''
        if state.patient_list==None:
            return False
        for patient in state.patient_list :
            if (patient.currWaitTime > self.labels[patient.labelID].maxWaitTime) or (patient.remainConsultTime != 0):
                return False
        return True

    def path_cost(self,cost,state1,action,state2):
        '''
        Returns the path cost of state s2, reached from state s1 by
        applying action a, knowing that the path cost of s1 is c.
        We consider the following cost associated to all the patients in the waiting room:
            C(P) = SUM(p in P) (p_cw)=^2 where p_cw is the patient waiting time
        '''
        '''
        state1Cost=0
        for patient in state1.patient_list:
            state1Cost+=patient.currWaitTime*patient.currWaitTime
        '''
        if state2.patient_list==None:
            return float('inf')
        state2Cost=0
        for patient in state2.patient_list:
            state2Cost+=patient.currWaitTime*patient.currWaitTime

        return state2Cost

    def load(self,file):
        '''
        Loads a problem from a (opened) file object f (see below for format specification)
        '''
        patient_list=list()
        doctor_assignments=list()
        self.inputfile = file
        return
        for line in file:
            line = line.rstrip()
            if not line:
                continue
            info=line.split()
            if (info[0]=='MD'):
                #dict with keys as doc_id and doctors as values
                self.doctor_dict[info[1]]=Doctor(info[1],float(info[2]))
            elif(info[0]=='PL'):
                self.labels[info[1]]=PatientLabel(int(info[2]),int(info[3]))
            elif(info[0]=='P'):
                patient_list.append(Patient(info[1],int(info[2])
                ,info[3],int(self.labels[info[3]].consult_time),
            self.labels[info[3]].maxWaitTime))
        self.numb_docs=len(self.doctor_dict.keys())
        self.initial=State(patient_list,0,doctor_assignments)


    def save(self,f):

        f.write('MD ')
        for line in self.inputfile:

             f.write(line.rstrip())
             f.write('  newline  ')

    def search(self,**kwargs):
        return True


    def heuristic(self,node):
        return 0

```

◄ Announcements

Jump to...

Report ►

VPL