Вештачка интелегенција

Домашна задача 1 – група 4

ПРОБЛЕМИ СО ПРЕБАРУВАЊЕ НА ПРОСТОРОТ НА СОСТОЈБИ И ЗАДОВОЛУВАЊЕ УСЛОВИ

Петар Атанасовски – 216052

1. **<u>Проблем со пребарување на простор на состојби - Размена на Пакмани</u>**
   a. **Дефинирајте минимална репрезентација на состојбата на проблемот користејќи математичка нотација по ваш избор за елементите на состојбата. Образложете го секој елемент на состојбата.**
      Имаме (M/2) жолти и (M/2) зелени пакмани, вкупно M пакмани. За секој пакман потребен ни е пар (x, y) каде x и y се координатите на тој пакман. Значи за целосна минимална репрезентација на состојба на проблемот потребни ни се **M парови (x, y).**
   b. **Колку најмногу состојби може да има во просторот на состојби за вашата дефиниција? Образложете го секој дел од вашето решение.**
      На таблата имаме (MxN) позиции и M пакмани. Секој пакман може да се најди во една од можните позиции. Значи **вкупниот број на состојби е $(MxN)^M$**.
   c. **Која е максималната вредност на факторот на разгранување (branching factor) за овој проблем? Образложете.**
      Максимален (branching factor) е максималниот број на деца на било кој јазол во едно дрво. За дадениот проблем еден пакман би можел да оди во 4 различни насоки и тоа лево и десно може да направи N - 1 чекори, а горе и долу M - 1 чекори. Значи еден пакман може од една состојба да се најде во (N-1 + M-1), односно (N + M - 2) други состојби. Истото ова можат да го направат сите M пакмани, од каде што следи дека од една состојба можни се M(M + N - 2) нови состојби. Значи **максимален branching factor е M(M + N - 2).**
   d. **Напишете ги почетната и целната состојба дадени на сликите според вашата дефиниција**
      Според мојата дефиниција (прашање а)):
      Почетна состојба: (((0, 2), (0, 3)), ((4, 0), (4, 1)))
      Целна состојба: (((4, 0), 4, 1)), ((0, 2), (0, 3)))
   e. **Ако проблемот се наоѓа во некоја произволна состојба кои акции се дозволени (легални)? Образложете го одговорот, користејќи ја вашата дефиниција за состојбата. Ве охрабруваме да користите програмски или псевдо-код во вашето објаснување.**
      Легални акции на пакман:
      I. Дали позицијата е надвор од границите на табелата
      II. Дали на таа позиција има друг пакман

Пример код за движење горе на жолт пакман:

```
for yp in yellow_pacmans:
    x, y = yp
    # up
    i = 1
    while y + i < self.M:
        new_pacman = (x, y + i)
        if new_pacman in yellow_pacmans or new_pacman in green_pacmans:
            break
        new_yellow_pacmans = list(state[0])
        new_yellow_pacmans.remove(yp)
        new_yellow_pacmans.append(new_pacman)
        new_state = (tuple(new_yellow_pacmans), tuple(green_pacmans))
        successors[
            'Yellow pacman from (' + str(x) + ', ' + str(y) + ') - ' + str(i) + ' positions up'] = new_state
        i += 1
```

(Со “y + I < self.M“ се проверува дали при движење на горе излегуваме од границите на табелата. Дозволени се движења до рамките на табелата. Со “if new_pacman in yellow_pacmans or new_pacman in green_pacmans:“ се проверува дали има друг пакман на позицијата. Доколку има не ја додаваме новата состојба и престануваме да одиме во истиот правец (да не го прескокнеме постоечкиот пакман))

f. **Дефинирајте самите некоја можна нетривијална евристика за поедноставена верзија на проблемот во која имате само еден Пакман на ролери на површината, на пример жолт Пакман кој се наоѓа во горната половина на првата колона и треба да стигне во долната половина на последната колона. Докажете дека вашата евристика е допустлива.**

Допусна евристика е евристика која дава оптимистичка вредност за бројот на чекори, секогаш помала или еднаква од вистинскиот број на чекори потребен да се стигне до конечното решение. За еден пакман ваква евристика може да се дефинира така што:

I. Да дава вредност 0 доколку пакманот е на точна позиција
II. Да дава вредност 1 доколку пакманот се наоѓа во иста редица со целната состојба (има иста y координата). Бидејќи кога сме на во иста редица можиме да стигнеме до целта во 1 чекор (доколку нема други пакмани на патот)
III. Да дава вредност 1 доколку пакманот се наоѓа во иста колона со целната состојба (има иста x координата). (Објаснување од ii.)
IV. Во сите останати случаи да враќа 2. Бидејќи од било која позиција до било која друга позиција на табелата можеме да стигнеме во максимум 2 чекори (доколку нема други пакмани на патот)

Имплементацијата во код за дадениот проблем во кој имаме M / 2 целни состојби за еден пакман односно состојбите (N - 1, 0), (N - 1, 1), ... (N - 1, M / 2 - 1) би изгледала вака: (ова е пример за жолт пакман):

```
def h_pacman_yellow(self, pacman):
    if pacman in self.yellow_goal:
        return 0
    x, y = pacman
    if 0 <= y < self.M / 2 or x == self.N - 1:
        return 1
    else:
        return 2
```

(self.yellow_goal е сет од сите можни целни состојби на жолт пакман (бидејќи не е битен распоредот на пакманите, само тие да се на точните целни позиции))

g. Нека со h_i е означена вашата евристика дефинирана под (ѓ) која се однесува на i-тиот Пакман на ролери во оригиналниот проблем со N Пакмани на ролери. Која од следните евристики е допустлива? Образложете за секоја евристика.

Евристики:

I. H_a: **Допуслива**, Во еден чекор се движи само еден пакман, што значи дека апсолутната средина на евристиките на сите пакмани е секогаш помала од вистинскиот број на чекори, а доклку сите пакмани се на своите позиции имаме 0/N што е 0

II. H_b: **Допуслива**, Во еден чекор се движи само еден пакман, значи треба да се придвижат сите N пакмани и секој стига до целта за минимална вредност од h_i. Значи и сумата од сите h_i е секогаш помала или еднаква од вистинската вредност на број на чекори.

III. H_c: **Допуслива**, Доколку сумата од сите h_i е допуслива, се подразбира дека и максималната евристика е исто така допуслива.

IV. H_d: **Не допуслива**, Долку сите пакмани се на своите целни позиции освен еден, оваа евристика би дала број поголем од вистинската вредност на број на чекори што ја прави оваа евристика не допуслива.

V. H_e: **Тривијална**, Доколку еден пакман е на точна позиција оваа евристика враќа вредност 0, но тоа е тривијално бидејќи другите пакмани можеби сеуште не се на своите целни позиции.

VI. H_f: **Тривијална**, Доколку еден пакман е на точна позиција оваа евристика враќа вредност N*0 = 0, но тоа е тривијално бидејќи другите пакмани можеби сеуште не се на своите целни позиции.

Во мојот код употребена е евристиката H_b, која сметам дека дава најдобра апроксимација за вистинската вресност.

```
def h(self, node):
    suma = 0
    for pacman in node.state[0]:
        suma += self.h_pacman_yellow(pacman)
    for pacman in node.state[1]:
        suma += self.h_pacman_green(pacman)
    return suma
```

(Во state[0] се жолтите пакмани, а во state[1] се зелените. Преку овај код се прави сума од евристиките (h_i од прашање f) на сите пакмани)

h. Кој од следните алгоритми за пребарување ќе гарантираат најбрза замена на Пакман играчите (оптималност)? Образложете за секој алгоритам. Доколку имате избор, кој од понудените алгоритми би го избрале како најдобар за проблемот? Образложете го вашиот избор.

Алгоритми:

I. **DFS** – овој алгоритам е неинформиран, доста брз и пребарува низ дрвото на состојби прво по длабочина. Ни дава резултат но тој резултат не е секогаш оптималното решение. Може да се користи во средина со големи вредности за M и N.

II. **BFS** – овој алгоритам е неинформиран алгоритам кој дава точно и оптимално решение. Го пребарува дрвото прво по широчина. Проблем е тоа што во нашиот случај имаме голем branching factor па овај алгоритам би работел бавно и би имал доста лоши перформанси.

III. **UCS** – овој алгоритам е неинформиран, оптимален, се движи прво по патека со најниска цена од можните понудени. Негативна страна е тоа што не знае каде е целта, како и за BFS, DFS. Пред да преминеме во друга состојба овај алгоритам во предвид ја зема цената на патот до сега и од тие цени ја избира минималната.

IV. **A*** – овој алгоритам е информиран алгоритам и дава оптимално решение. За да го користиме овој алгоритам потребно е да имаме некоја допуслива евристика. Пред да премине во друга состојба овој алгоритам во предвид ги зема цената на патот плус евристиката. Тој е доста брз доколку имаме имплементирано соодветна евристика.

Јас во овој случај би го употребил A* алгоритмот при што би посветил доволно време за развивање на добра евристика. Со тоа би имал гарантиран оптимален резултат. Од друга страна ако алгоритмот треба да работи во голема средина и приоритет ми е брзо да дојдам до решение, без оглед дали тоа е оптимално, би употребил DFS или Greedy (Алчен) алгоритам.

2. Проблеми кои исполнуваат услови - Аеромитинг

a. Формално дефинирајте го проблемот како проблем на исполнување услови (Constraint Satisfaction Problem – CSP).

I. Променливи:

1. Авион на претседателот (president_plane)
2. Група борбени авиони (fighter_jets)
3. Нов хеликоптер (new_ helicopter)
4. Модел на авион од Втората светска војна (ww2_plane)

II. Домени: {P1_termin1, P2_termin1, P1_termin2, P2_termin2, P1_termin3, P2_termin3, P1_termin4, P1_termin4}:

```
variables = ["president_jet", "new_helicopter", "ww2_jet", "fighter_jets"]
domains = ["P1_1", "P1_2", "P1_3", "P1_4", "P2_1", "P2_2", "P2_3", "P2_4"]
```

(Дополнително, за појасно дефинирање на условите во останатиот дел од овај документ, преку **term(variable)** го означувам терминот во кој слетува дадениот авион, без разлика на која писта. Пример: term(president_plane) се однесува на терминот на слетување на авионот на претседателот)

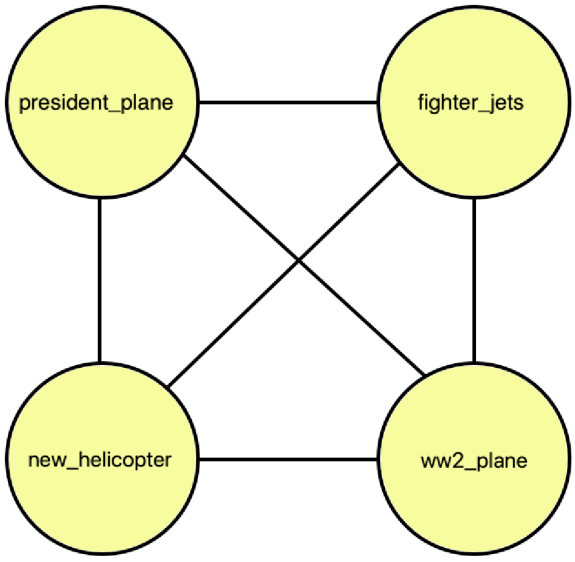
III. Услови:

1. Авионот на претседателот на писта 1:
   a. president_plane = {P1_termin1, P1_termin2, P1_termin3, P1_termin4}
2. Авионот ќе биде затворен за слетувања во периодот пред да пристигне претседателот. Групата борбени авиони е изземена од оваа забрана:
   a. | term(president_plane) - term(new_ helicopter)| > 1
   b. | term(president_plane) – term(ww2_plane) | > 1
3. Групата борбени авиони кои слетуваат како формација има потреба од двете писти за нивно слетување: Нема никој друг на другата писта во аеродромот додека слетуваат борбените авиони – повеќе бинарни услови
   a. term(president_plane) != term(fighter_jets)
   b. term(new_ helicopter) != term(fighter_jets)
   c. term(ww2_plane) != term(fighter_jets)
4. Новиот хеликоптер кој ќе биде претставен денес мора да слета пред слетувањето на претседателот.
   a. term(new_ helicopter) < term(president_plane)
5. Новиот хеликоптер мора да слета на писта P2:
   a. new_ helicopter = { P2_termin1, P2_termin2, P2_termin3, P2_termin4}
6. Моделот на авион од Втората светска војна не смее да слета истовремено или после слетувањето на новиот хеликоптер:
   a. term(ww2_plane) < term(new_ helicopter)

Имплементација на условите во код:

```
problem.addConstraint(AllDifferentConstraint())
# Unary
problem.addConstraint(lambda x: str(x).split("_")[0] == "P1", ['president_jet'])
problem.addConstraint(lambda x: str(x).split("_")[0] == "P2", ['new_helicopter'])
# Binary
problem.addConstraint(lambda x, y: abs(int(str(x).split("_")[1]) - int(str(y).split("_")[1])) > 1,
                      ['president_jet', 'new_helicopter'])
problem.addConstraint(lambda x, y: abs(int(str(x).split("_")[1]) - int(str(y).split("_")[1])) > 1,
                      ['president_jet', 'ww2_jet'])
problem.addConstraint(fighterJetsConstraint, variables)
problem.addConstraint(lambda x, y: int(str(x).split("_")[1]) < int(str(y).split("_")[1]),
                      ['new_helicopter', 'president_jet'])
problem.addConstraint(lambda x, y: int(str(x).split("_")[1]) < int(str(y).split("_")[1]),
                      ['ww2_jet', 'new_helicopter'])
```

b. Нацртајте го графот на ограничувања (услови) за проблемот.



I. new_helicopter <–> ww2_plane – унарен услов
II. new_helicopter <–> president plane - унарен услов
III. сите поврзани со fighter_jets бидејќи им се потребни двете писти
IV. new_helicopter и ww2_plane поврзани со president_plane поради забраната за слетување пред, за време на и по авионот на претседателот

c. Започнувате да го решавате проблемот со исполнување на унарните услови. Унарните услови се однесуваат на дозволени и недозволени вредности за променливите во проблемот. За домените на секоја променлива да се провери кои вредности преостануваат по примена на сите унарни услови од проблемот. Да се прикажат вредностите кои преостануваат во доменот на секоја променлива!

I. president_plane = { P1_termin1, P1_termin2, P1_termin3, P1_termin4}
II. new_ helicopter = { P2_termin1, P2_termin2, P2_termin3, P2_termin4}

III. fighter_jets = { P1_termin1, P2_termin1, P1_termin2, P2_termin2, P1_termin3, P2_termin3, P1_termin4, P2_termin4}

IV. ww2_plane = { P1_termin1, P2_termin1, P1_termin2, P2_termin2, P1_termin3, P2_termin3, P1_termin4, P2_termin4}

	president_plane	fighter_jets	new_ helicopter	ww2_plane
P1_termin1			X	
P2_termin1	X			
P1_termin2			X	
P2_termin2	X			
P1_termin3			X	
P2_termin3	X			
P1_termin4			X	
P2_termin4	X			

(Како би изгледала табелата по задавање на унарните услови, со X се означени полињата чија вредност не може да ја прими дадената променлива)

d. Пред да се направи било каква додела на променливите, да се направи пропагација на условите со методот за проверка на конзистентноста на целиот проблем т.е. проверка на конзистентност на сите ребра од графот (arc consistency enforcing). За секоја направена проверка да се идентификува реброто чија конзистентност се проверува и прикажат промените во доменот на засегнатите променливи! (Не е потребно да се прикажува секоја итерација на постапката, може да ги прикажете само проверките за конзистентност на ребра каде се случуваат промени. Доколку дискусијата на постапката за повеќе ребра е иста/слична, дискусијата може да се однесува на група ребра.)

Приказ на чекорите при пропагација на условите (arc consistency enforcing)

1. term(ww2_plane) < term(new_helicopter) => **term(ww2) != 4 and term(new_helicopter) != 1**
2. term(new_helicopter) < term(president_plane) => **term(new_helicopter) != 4 and term(president_plane) != 1**
3. term(new_helicopter) !=4 and term(ww2_plane) < term(new_helicopter) => **term(ww2_plane) != 3**
4. |term(new_helicopter) < term(president_plane)| > 1 => **term(new_ helicopter) != 3**
5. term(new_helicopter) = 2 and (term(ww2_plane) < term(new_helicopter)) => **term(ww2_plane) != 2**
6. term(ww2_plane) = 1 => **term(fighter_jets) != 1**
7. term(new_helicopter) = 2 => **term(fighter_jets) != 2**
8. term(president_plane) > term(new_helicopter) => **term(president_plane) != 2**
9. term(new_helicopter) = 2 and |term(new_helicopter < term(president_plane)| > 1 => **term(president_plane) != 3**

(Со задебелен текст претставени се заклучоците до кои доаѓаме за да ги избришеме вредностите од домените)

По пропагирање на условите, табелата на можни вредности за секоја од променливите изгледа вака:

	president_plane	fighter_jets	new_ helicopter	ww2_plane
P1_termin1	X	X	X	
P2_termin1	X	X	X	
P1_termin2	X	X	X	X
P2_termin2	X	X		X
P1_termin3	X		X	X
P2_termin3	X		X	X
P1_termin4		X	X	X
P2_termin4	X	X	X	X

e. **Применувајќи евристика за определување на следна променлива за додела на вредност, евристика за избор на вредност и проверка нанапред (forward checking) најдете едно решение за проблемот од примерот! За секое направено доделување да се образложи која евристика е искористена и како тоа доделување се одразува во проверката нана**

Прва евристика која ја употребуваме е **MRV(Minimum Remaining Values),** тука имаме tie-brake на president_plane со new_helicopter и fighter_jets (бидејќи на fighter_jets потребни им се двете писти). Потоа користиме **LCS (Least Cinstraining Value)** каде повторно има tie-brake. Од табелата погоре можеме да заклучиме дека само ww2_plane може да биде на писта 1 или 2 во термин 1, за сите останати променливи имаме само по 1 можна вредност во доменот.

Дадениете задачи се целосно решени во python, и нивното решение може да се види на следниот линк:
https://github.com/AtanasovskiPetar/FINKI/tree/master/VI%20(Artificial%20Intelligence)/Domashni/Domashna_1/Homework_Codes