Вештачка интелигенција

Домашна задача 1 – Група 2

Владимир Христовски – 223030

1. Токсичен Пакман

А:

- Проблемот ќе го анализираме со тоа што дозволените позиции на кои секој еден Пакман може да се движи ќе го претставиме со матрица со M колони и N редици

M – колони до кои може Пакман да се движи, y оска

N – редици до кои може Пакман да се движи, x оска

k – број на Пакман играчи

$(M * N)^k * (M * N)^{2k}$

- Имаме k двојки од вредности $\big((X_1, Y_1), (X_2, Y_2), \dots\ (X_k, Y_k)\big)$, каде (X_k, Y_k) претставуваат X и Y координати на пакманите

- Бидејќи едно поле останува отровно 2 чекори, соодветно ќе имаме

$k * \ ((X_1, Y_1), (X_2, Y_2))$ каде X и Y претставуваат координати на отровните полиња

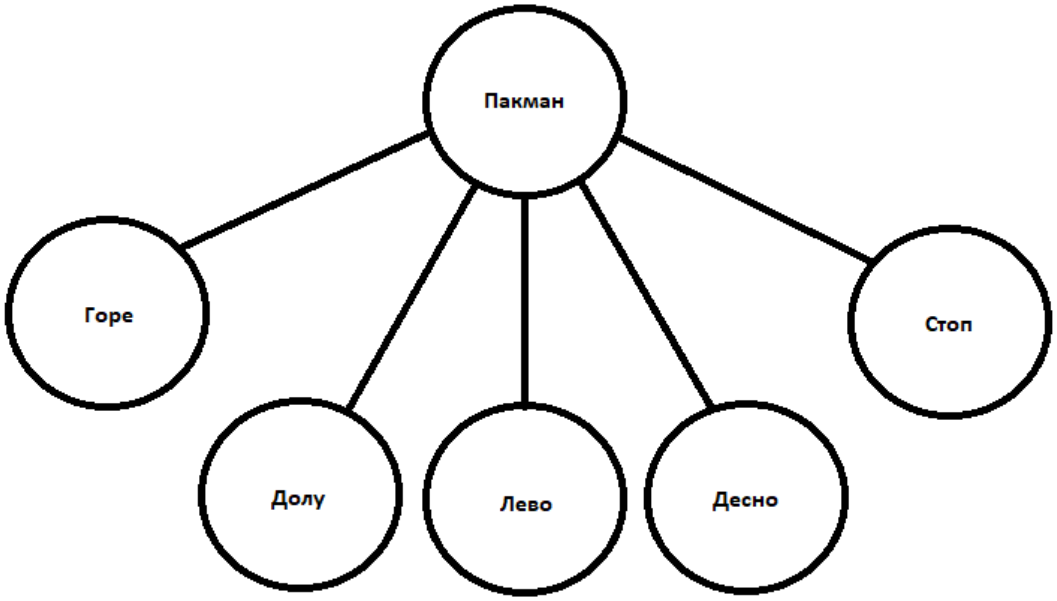
Б:

Најголемиот број на состојби може да се добие со замена на соодветните величини на променливите од минималната репрезентација на проблемот

$(M * N)^k * (M * N)^{2k}$

Каде M претставуваат колоните на кои може Пакман да се движи, N претставуваат редовите по кои Пакман може да се движи и k претставува број на пакмани

В:



Максималната вредност на факторот на разгранување за еден Пакман е 5, бидејќи секој Пакман во идеални услови, каде тој е осамен во огромен дозволен простор и нема друг Пакман или sид или отровна трага околу него, би имал 5 акции кои би можел да ги преземи и тоа : придвижи се горе, придвижи се долу, придвижи се лево, придвижи се десно или остани на тековната позиција. Во нашиот случај бидејќи имаме k пакмани кои се движат истовремено, факторот на разгранување би бил 5^k

Г:

Почетната состојба јас ја дефинирам како: позиција на првиот Пакман, позиции на токсични траги на прв Пакман, позиција на втор Пакман, позиции на токсични траги на вториот Пакман. Оваа состојба би ми била state при решавање бидејќи овие вредности се менуваат динамички

(0, 1), (), (0, 4), ()

Додека вредностите на sидовите не се менуваат, тие би ми биле сместени во променлива self.walls чија содржина би изгледала вака:

(2, 2), (3, 2), (3, 3), (3, 4), (4, 2)

Целната состојба би изгледала вака:

(6, 4)

Тоа прикажано во код би изгледало вака:

```
class Toxic_Pacman(Problem):

    def __init__(self, pacman_1, pacman_2, hospital, walls, width, height):
        super().__init__((pacman_1, (), pacman_2, ()))
        self.width = width
        self.height = height
        self.hospital = hospital
        self.walls = walls

    def goal_test(self, state):
        return state[0] == state[2] == self.hospital

if __name__ == '__main__':
    width = 7
    height = 5
    pacman_1 = (0, 4)
    pacman_2 = (0, 1)
    hospital = (6, 4)
    walls = ((2, 2), (3, 2), (3, 3), (3, 4), (4, 2))

    problem = Toxic_Pacman(pacman_1, pacman_2, hospital, walls, width,
height)
```

Д:

Еден Пакман има избор од 5 акции кои може да ги преземе и тоа: придвижи се горе, придвижи се долу, придвижи се лево, придвижи се десно или остани на тековната позиција. Во зависност од тоа дали Пакманот после акцијата се наоѓа во легална состојба, која подразбира истиот да не биде на исти координати како и некои од ѕидовите или истиот да не биде на исти координати како и некој друг Пакман (со исклучок на целната состојба) или истиот да не биде на исти координати со неговата отровна трага или отровната трага на некој друг Пакман

За задачата од примерот кодот би изгледал вака:

```
def check_valid(self, state):
    pacman_1 = state[0]
    pacman_1_trail = state[1]
    pacman_2 = state[2]
    pacman_2_trail = state[3]
    if pacman_1 == pacman_2 == self.hospital:
        #Доколку пакман 1 е на иста позиција со пакман 2 и воедно двата се на
позицијата на болницата врати True
        return True
```

```
elif pacman_1 in self.walls or pacman_2 in self.walls or pacman_1 == pacman_2
or pacman_1[0] < 0 or pacman_1[
        0] >= self.width or pacman_1[1] < 0 or pacman_1[1] >= self.height or
pacman_2[0] < 0 or pacman_2[
        0] >= self.width or pacman_2[1] < 0 or pacman_2[
        1] >= self.height or pacman_1 in pacman_1_trail or pacman_1 in
pacman_2_trail or pacman_2 in pacman_2_trail or pacman_2 in pacman_1_trail:
        #Доколку пакман 1 или пакман 2 се најдат на иста позиција или надвор
од дозволениот опсег или во ѕид
        # или во сопствената токсична трага или во токсичната трага на
другиот пакман да врати False
        return False
    else:
        #Во спротивно врати True
        return True

def move_pacman1(self, state, direction, action):
    pacman_1 = list(state[0])
    pacman_1_trail = list(state[1])

    pacman_1[0] += direction[0]
    pacman_1[1] += direction[1]

    if action == "Stop":
        #Доколку акцијата е да остани во место и должината на неговата
токсична трага е поголема од 0, намали ја токсичната трага за 1
        if len(pacman_1_trail) > 0:
            pacman_1_trail = pacman_1_trail[1:]
    else:
        #Во спротивно
        if len(pacman_1_trail) < 3:
            #Доколку токсичната трага е со должина помала од 3, зголеми ја
трагата за претходната состојба на пакман
            pacman_1_trail.append(state[0])
        elif len(pacman_1_trail) == 3:
            #Доколку токсичната трага е со должина 3, зголеми ја трагата за
претходната состојба на пакман, па скрати намли ја токсичната трага за 1
            pacman_1_trail.append(state[0])
            pacman_1_trail = pacman_1_trail[1:]

    check_state = (tuple(pacman_1), tuple(pacman_1_trail), state[2],
state[3])

    if self.check_valid(check_state):
        #Проверка дали позициите на пакман се валидни
        return check_state
    else:
        return None

def move_pacman2(self, state, direction, action):
    pacman_2 = list(state[2])
    pacman_2_trail = list(state[3])

    pacman_2[0] += direction[0]
    pacman_2[1] += direction[1]
```

```
if action == "Stop":
        #Доколку акцијата е да остани во место и должината на неговата
токсична трага е поголема од 0, намали ја токсичната трага за 1
        if len(pacman_2_trail) > 0:
            pacman_2_trail = pacman_2_trail[1:]
    else:
        if len(pacman_2_trail) < 3:
            #Доколку токсичната трага е со должина помала од 3, зголеми ја
трагата за претходната состојба на пакман
            pacman_2_trail.append(state[2])
        elif len(pacman_2_trail) == 3:
            #Доколку токсичната трага е со должина 3, зголеми ја трагата за
претходната состојба на пакман, па скрати намли ја токсичната трага за 1
            pacman_2_trail.append(state[2])
            pacman_2_trail = pacman_2_trail[1:]

    check_state = (state[0], state[1], tuple(pacman_2),
tuple(pacman_2_trail))

    if self.check_valid(check_state):
        #Проверка дали позициите на пакман се валидни
        return check_state
    else:
        return None

def successor(self, state):
    neighbours = dict()

    actions = ("Gore", "Dolu", "Levo", "Desno", "Stop")
    #Дозволени акции за движење

    directions = ((0, +1), (0, -1), (-1, 0), (+1, 0), (0, 0))
    #Дозволени насоки за движење

    for action1, direction1 in zip(actions, directions):
        res1 = self.move_pacman1(state, direction1, action1)
        #Прво го придвижуваме пакман 1
        if res1 is not None:
            #Доколку неговата позиција е валидна продолжуваме тука
            for action2, direction2 in zip(actions, directions):
                res2 = self.move_pacman2(res1, direction2, action2)
                #Потоа го придвижуваме пакман 2
                if res2 is not None:
                    #Конечно, доколку позициите на двата пакмани е валидна да
се додаде во речникот со акции
                    neighbours["Pacman_1 : " + action1 + ", Pacman_2 : " +
action2] = res2

    return neighbours
```

Ѓ:

За дефиницијата на нетривијалната евристика за еден Пакман можеме да искористиме Менхетен растојание од Пакманот до болницата кое би било 0 кога Пакманот ќе се наоѓа на координатите на болницата, исто така оваа евристика е допустлива бидејќи вредноста на евристиката ќе биде секогаш помала или еднаква на реалната вредност, со оглед на тоа дека Пакман при патувањето ќе се соочи со препреки, но дури и да не се соочи реалното растојание ќе биде еднакво на растојанието од евристиката

| X координата на Пакман – X координата на болница | + | Y координата на Пакман – Y координата на болница |

Тоа во код би изгледало вака :

```
def h(self, node):
    state = node.state
    pacman = state[0]
    return abs(pacman[0] - self.hospital[0]) + abs(pacman[1] -
self.hospital[1])
```

Е:

$$h_a : \frac{\sum_{i=1}^{k} h_i}{k}$$

Оваа евристика се заснова врз средната вредност од преостанатиот пат до целта, односно збир на сите евристики, во овој случај Менхетен растојанија поделен со бројот на пакмани. Според мене оваа евристика е допустлива бидејќи со оваа евристика вистинското растојание од барем еден Пакман до болницата би било поголемо или еднакво од растојанието на евристиката

$$h_b : \sum_{i=1}^{k} h_i$$

Оваа евристика се заснова врз вкупниот преостанат пат до целта, односно збир од сите Менхетен растојанија. Според мене оваа евристика е допустлива бидејќи со оваа евристика збирот на вистинските растојанија од сите пакмани до болницата би бил еднаков или поголем од растојанието од евристиката со оглед на тоа што пакманите ќе се соочат со препреки при патувањето

$$h_c: \max_{1 \le i \le k} h_i$$

Оваа евристика се заснова врз евристиката на Пакманот кој што е најоддалечен од болницата. Според мене оваа евристика е допустлива бидејќи со оваа евристика само најоддалечениот Пакман од болницата би правел правилни потези и неговото растојание до болницата би било еднакво или поголемо од растојанието на евристиката со оглед на тоа што тој би можел да наиде на препреки при патувањето

$$h_d: k * \max_{1 \le i \le k} h_i$$

Оваа евристика се заснова врз збирот на сите евристики на пакманите при што сите пакмани ја имаат евриситиката на најоддалечениот Пакман од болницата. Според мене оваа евристика не е допустлива бидејќи е премногу песимистична, односно евристиката секогаш ќе дава вредност поголема или еднаква (доколку има преостанато само еден пакман) на реалната вредност, односно оддалеченост

$$h_e: \min_{1 \le i \le k} h_i$$

Оваа евристика се заснова врз евристиката на Пакманот кој што е најблиску до болницата. Според мене оваа евристика е допустлива но би станала тривијална во моментот кога еден Пакман ќе стигне до болницата, бидејќи евристиката ќе има вредност 0

$$h_f: k * \min_{1 \le i \le k} h_i$$

Оваа евристика се заснова врз збирот на сите евристики на пакманите при што сите пакмани ја имаат евристиката на најблискиот Пакман до болницата. Според мене оваа евристика е допустлива но би станала тривијална кога еден Пакман би стигнал до болницата со што евристиката ќе има вредност 0

Ж:

DFS (Depth First Search) – со користење на овој алгоритам би се пребарувало по длабочина почнувајќи од лево кон десно. Што полево се наоѓа решението во дрвото тоа би се нашло побрзо, но не значи дека тоа е оптимално решение, односно најкратко

BFS (Breadth First Search) – со користење на овој алгоритам ќе биде пронајдено најоптималното решение, но пребарувањето би можело да одземе многу време бидејќи се пребарува по широчина

UCS (Uniform Cost Search) – со користење на овој алгоритам ќе биде пронајден најкраткиот пат со тоа што сите акции имаат иста цена (цената е 1) и затоа овој алгоритам би се однесувал исто како и BFS, односно ќе биде пронајдено најоптималното решение

A* - со користење на овој алгоритам ќе биде пронајден најкраткиот пат за најкратко време ако користиме допустлива евристика. Со таква евристика пребарувањето би било насочено кон целта и сите непотребни состојби би биле избегнати

- Јас би го избрал алгоритмот A* со допустлива евристика, бидејќи пакманите така би стигнале најбрзо до болницата, без да истражуваат состојби кои ќе ги оддалечат од истата беспотребно

2. Какурасу

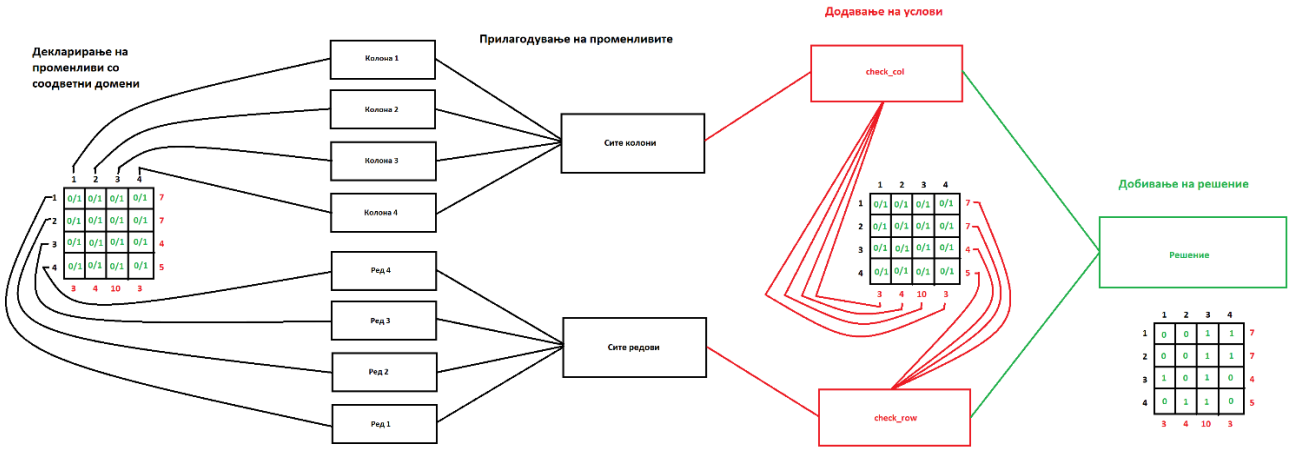
A:

Во моето решение на задачата јас променливите ги зададов како сите полиња од табелата. Секое поле има домен на вредности од 0 или 1 соодветно. Ограничувањата кои се присутни во овој пример се сумите од редиците да изнесуваат 7, 7, 4, 5, соодветно, почнувајќи од првата редица, односно од горе па надолу по табелата, следно имаме и ограничување на сумите од колоните кои треба да изнесуваат 3, 4, 10, 3, соодветно, почнувајќи од првата колона, односно од лево па надесно по табелата. Соодветно мојот код од оваа задача каде го дефинирам проблемот изгледа вака:

```
if __name__ == '__main__':
    problem = Problem(BacktrackingSolver())
    variables = []

    for i in range(1, 5):
        for j in range(1, 5):
            variables.append((i, j))
            problem.addVariable((i, j), [0, 1])
            #Додавање на сите полиња од табелата во променливата variables
```

Б:




```python
def check_col(*items):
    col_constraints = [3, 4, 10, 3]

    sum = 0
    num = 1
    con = 0
    for i in range(len(items)):
        if items[i] == 1:
            sum += num
        num += 1
        if num == 5:
            if sum != col_constraints[con]:
                return False
            sum = 0
            num = 1
            con += 1
    return True


def check_row(*items):
    row_constraints = [7, 7, 4, 5]

    sum = 0
    num = 1
    con = 0
    for i in range(len(items)):
        if items[i] == 1:
            sum += num
        num += 1
        if num == 5:
            if sum != row_constraints[con]:
                return False
            sum = 0
            num = 1
            con += 1
    return True
```

```
if __name__ == '__main__':
    row_check = []
    for row in range(1, 5):
        for col in range(1, 5):
            row_check.append((row, col))
    problem.addConstraint(check_row, row_check)

    col_check = []
    for col in range(1, 5):
        for row in range(1, 5):
            col_check.append((row, col))
    problem.addConstraint(check_col, col_check)
```

В:

Бидејќи во задачава немаме унарни услови конкретно дефинирани за некое поле, табелата со променливи и соодветните домени изгледа вака:

Променлива	i_1,j_1	i_1,j_2	i_1,j_3	i_1,j_4	i_2,j_1	i_2,j_2	i_2,j_3	i_2,j_4	i_3,j_1	i_3,j_2	i_3,j_3	i_3,j_4	i_4,j_1	i_4,j_2	i_4,j_3	i_4,j_4
Домен	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1	0, 1

Но ние можеме да поставиме глобални услови за полињата од колона 3, бидејќи збирот на таа колона мора да биде 10, според претходно поставениот услов за суми на колони и редови, па според тоа сите 4 полиња од колона 3 мораат да ја имаат вредноста 1, односно да бидат обоени. Слично може да се наведе и за другите полиња според нивните услови за суми на колоните и редовите со цел да се дојде до комплетно решение. Ако доделуваме глобални услови, тогаш табелата би изгледала вака :

Променлива	i_1,j_1	i_1,j_2	i_1,j_3	i_1,j_4	i_2,j_1	i_2,j_2	i_2,j_3	i_2,j_4	i_3,j_1	i_3,j_2	i_3,j_3	i_3,j_4	i_4,j_1	i_4,j_2	i_4,j_3	i_4,j_4
Домен	0, 1	0, 1	1	0, 1	0, 1	0, 1	1	0, 1	0, 1	0, 1	1	0, 1	0, 1	0, 1	1	0, 1

Г:

Во конкретниов пример, како што веќе објаснив погоре дека имаме можност да доделиме глобални услови, тоа всушност е и проверка на конзистентност за тие специфични ребра, бидејќи ако барем едно од тие ребра односно барем едно поле во колоната 3 ако се избере да не биде обоено, односно да има вредност 0, тогаш во тој случај нема да се дојде до комплетно решение, бидејќи сумата за таа колона нема да изнесува 10. Слично би било и за другите ребра како на пример за колона 1, за исполнување на условот на сума на колона во таа колона може да бидат обоени или само третото поле од горе надолу или само првото и второто поле гледано од горе надолу за да биде исполнет условот за сума, но никако не смее да биде обоено

четвртото поле гледано од горе кон долу, со оглед на тоа што условот на таа колона е да има сума 3, истото важи и за колона 4 каде исто така сумата на колоната мора да изнесува 3. После проверката табелата со променливи и домени би изгледала вака:

Променлива	i_1, j_1	i_1, j_2	i_1, j_3	i_1, j_4	i_2, j_1	i_2, j_2	i_2, j_3	i_2, j_4	i_3, j_1	i_3, j_2	i_3, j_3	i_3, j_4	i_4, j_1	i_4, j_2	i_4, j_3	i_4, j_4
Домен	0, 1	0, 1	1	0, 1	0, 1	0, 1	1	0, 1	0, 1	0, 1	1	0, 1	0	0, 1	1	0

Д:

Табелата после проверка на козистентност изгледа вака:

Променлива	i_1, j_1	i_1, j_2	i_1, j_3	i_1, j_4	i_2, j_1	i_2, j_2	i_2, j_3	i_2, j_4	i_3, j_1	i_3, j_2	i_3, j_3	i_3, j_4	i_4, j_1	i_4, j_2	i_4, j_3	i_4, j_4
Домен	0, 1	0, 1	1	0, 1	0, 1	0, 1	1	0, 1	0, 1	0, 1	1	0, 1	0	0, 1	1	0

За да дојдеме до решение на овој проблем ние мораме да го искористиме алгоритамот за враќање наназад со проверка нанапред (backtracking algorithm with forward checking) се со цел да најдеме комплетно решение на проблемот

Променлива	i_1, j_1	i_1, j_2	i_1, j_3	i_1, j_4	i_2, j_1	i_2, j_2	i_2, j_3	i_2, j_4	i_3, j_1	i_3, j_2	i_3, j_3	i_3, j_4	i_4, j_1	i_4, j_2	i_4, j_3	i_4, j_4
Домен	1	0, 1	1	0, 1	1	0, 1	1	0, 1	0	0, 1	1	0, 1	0	0, 1	1	0

Доделени вредности за колона 1

Променлива	i_1, j_1	i_1, j_2	i_1, j_3	i_1, j_4	i_2, j_1	i_2, j_2	i_2, j_3	i_2, j_4	i_3, j_1	i_3, j_2	i_3, j_3	i_3, j_4	i_4, j_1	i_4, j_2	i_4, j_3	i_4, j_4
Домен	1	1	1	0, 1	1	0	1	0, 1	0	1	1	0, 1	0	0	1	0

Доделени вредности за колона 2, при доделување на овие вредности можеме да заклучиме дека условот за сума на ред 4 е невалиден, па со помош на алгоритамот за враќање наназад со проверка нанапред (backtracking algorithm with forward checking) се враќаме на претходната постапка и доделуваме други вредности

Променлива	i_1, j_1	i_1, j_2	i_1, j_3	i_1, j_4	i_2, j_1	i_2, j_2	i_2, j_3	i_2, j_4	i_3, j_1	i_3, j_2	i_3, j_3	i_3, j_4	i_4, j_1	i_4, j_2	i_4, j_3	i_4, j_4
Домен	1	0	1	0, 1	1	0	1	0, 1	0	0	1	0, 1	0	1	1	0

Засега со овие вредности сите услови за суми на колони и редови се исполнети, но при проверка на сумата за ред 1 и ред 2 можеме да заклучиме дека кај полињата i1, j4 и i2, j4 ако доделиме било која од вредностите сепак сумата за тие редови нема да биде 7, па повторно со алгоритамот за враќање наназад со проверка нанапред (backtracking algorithm with forward checking) се враќаме чекор назад за да го корегираме тоа

Променлива	i_1, j_1	i_1, j_2	i_1, j_3	i_1, j_4	i_2, j_1	i_2, j_2	i_2, j_3	i_2, j_4	i_3, j_1	i_3, j_2	i_3, j_3	i_3, j_4	i_4, j_1	i_4, j_2	i_4, j_3	i_4, j_4
Домен	0	0	1	0, 1	0	0	1	0, 1	1	0	1	0, 1	0	1	1	0

После оваа промена во колона 1 можеме да ги доделиме и останатите вредности

Променлива	i_1, j_1	i_1, j_2	i_1, j_3	i_1, j_4	i_2, j_1	i_2, j_2	i_2, j_3	i_2, j_4	i_3, j_1	i_3, j_2	i_3, j_3	i_3, j_4	i_4, j_1	i_4, j_2	i_4, j_3	i_4, j_4
Домен	0	0	1	1	0	0	1	1	1	0	1	0	0	1	1	0

Со што дојдовме до едно комплетно решение за овој проблем, каде сите услови за суми на колони и редови се исполнети.

При решавање на овој проблем искористени се евристиката за избор на вредност (Least Constraining Value - LCV) и евристиката за определување на следна променлива за додела на вредност (Minimum Constraining Values - MRV)