Задача №3 / 5

Задание 1. Прочитайте главу 7 из документа, присланного со второй задачей.

Задание 2. Реализуйте муравьиный алгоритм.

% муравьиный алгоритм

% -------------------------------------------------------------------------

tic

% clearvars -except cities

clearvars

% ------------------------------ИТЕРАЦИИ-----------------------------------

% кол-во итераций ( поколений )

age = 2000;

% кол-во муравьев в поколении

countage = 10;

% кол-во городов

n = 50;

% ------------------------------ПАРМЕТРЫ-----------------------------------

% альфа - коэффициент запаха, при 0 будем ориентироваться только на

% кратчайший путь

a = 1;

% бета - коэффициент расстояния, при 0 будем

% ориентироваться только на оставляемый запах

b = 2;

% коэффициент обновления, глобальное

e = 0.1;

% коэффициент обновления, локальное

p = 0.1;

% количество выпускаемых феромонов

Q = 1;

% баланс между лучшим городом и как в AS

q = 0.9;

% начальный феромон

ph = Q/(n\*2000);

% -------------------------------ПАМЯТЬ------------------------------------

% матрица расстояний

dist = zeros(n,n);

% матрица обратных расстояний

returndist = zeros(n,n);

% матрица маршрута муравьев в одном поколении

ROUTEant = zeros(countage,n);

% вектор расстояний муравьев в одном поколении

DISTant = zeros(countage,1);

% вектор лучших дистанций на каждой итерации

bestDistVec = zeros(age,1);

% лучший начальный маршрут

bestDIST = inf;

% оптимальные маршруты

ROUTE = zeros(1,n+1);

% перестановка городов без повторений ( для выхода муравьев )

RANDperm = randperm(n);

% матрица вероятностей

P = zeros(1,n);

% максимальное значение вероятности

val = zeros(1);

% присваем номер города

getcity = zeros(1);

% индекс максимального значения вероятности

indexP = zeros(1);

% максимальное

minDISTiterration = zeros(1);

% -------------------------------------------------------------------------

% генерация городов (x,y)

cities = rand(n,2)\*100;

% матрица начальных феромонов

tao = ph\*(ones(n,n));

tao(logical(eye(size(tao)))) = 0;

% создаем матрицу расстояний и матрицу обратных расстояний

for i = 1:n

for j = 1:n

% dist ( расстояния )

dist(i,j) = sqrt((cities(i,1) - cities(j,1))^2 + ...

(cities(i,2) - cities(j,2))^2);

% nn ( обратные расстояния )

if i ~= j

returndist(i,j) = 1/sqrt((cities(i,1) - cities(j,1))^2 + ...

(cities(i,2) - cities(j,2))^2);

end

end

end

% итерации

for iterration = 1:age

    % муравьи ( одно поколение)

    for k = 1:countage

    % \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* НАЧАЛЬНОЕ РАСПОЛОЖЕНИЕ МУРАВЬЕВ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    % выбирайте какой нужно

    % каждый муравей располагается случайно

      ROUTEant(k,1) = randi([1 n]);

    % с каждого города выходит один муравей ( без совпадений ), кол-во

    % городов и кол-во муравьев в поколении должны быть равны

%       ROUTEant(k,1) = RANDperm(k);

    % с конкретного города выходят все муравьи в данном случа с 1-ого

%       ROUTEant(k,1) = 1;

% тут маршрут первому поколению задаем либо произвольный, либо с каждого

% города разный, либо с одного города все, а следующее поколение выходит по

% концам первых

%     if iterration == 1

%       ROUTEant(k,1) = randi([1 n]);

% %       ROUTEant(k,1) = RANDperm(k);

% %     ROUTEant(k,1) = 1;

%     else

%       ROUTEant(k,1) = lastROUTEant(k);

%     end

    % \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    % путь каждого муравья, начиная со второго, так как первый выбран

    for s = 2:n

        % полуаем индекс выбранного города

        ir = ROUTEant(k,s-1);

        % вероятность посещения городов ( числитель ) , в числителе у нас

        % следующее: tao^a\*(1/S)^b

        % 1/S -это returndist.

        % поскольку данное значение будет повторяться (кол-во муравьев \* на

        % колонию \* кол-во городов) раз, то еще один цикл писать не выгодно,

        % скорость работы при таких вычислениях падает. Поэтому написал в

        % этом моменте векторно. На обычном языке будет так:

%         for c = 1:n

%             P(1,c) = tao(ir,c).^a \* returndist(ir,c).^b;

%         end

        P = tao(ir,:).^a .\* returndist(ir,:).^b;

        % получили числители (в формуле вероятности перехода к k-ому городу)

        % для n городов, однако в некоторых мы уже побывали, нужно исключить

        % их

        % проставляем нули в числитель туда, где уже были, чтобы

        % вероятность перехода была 0, следовательно в сумме знаменателя

        % формулы данный город учитываться не будет

        P(ROUTEant(k,1:s-1)) = 0;

        % смотрим в какой город осуществляется переход

        RANDONE = rand;

        if RANDONE <= q

            [val, getcity] = max(P);

        else

            % получаем вероятности перехода ( сумма строк должна быть = 1 )

            P = P ./ sum(P);

            getcity = find(cumsum(P) >= RANDONE, 1, 'first');

        end

        % присваем s-ый город в путь k-ому муравью

        ROUTEant(k,s) = getcity;

    end

    % получаем маршрут k-ого муравья

    ROUTE = [ROUTEant(k,1:end),ROUTEant(k,1)];

    % сброс длины

    S = 0;

    % вычисляем маршрут k-ого муравья

    for i = 1:n

        S = S + dist(ROUTE(i),ROUTE(i+1));

    end

    % путь k-ого муравья, массив дистанций k-ых муравьев age-ого поколения

    DISTant(k) = S;

    % присваевыем лучший маршрут и S

    if DISTant(k) < bestDIST

        bestDIST = DISTant(k);

        bestROUTE = ROUTEant(k,[1:end,1]);

        iter = iterration;

    end

    % вектор "последних" городов k-ых муравьев ( выбирается для старта

    % муравьев нового поколения с тех городов, где закончили путь

    % предыдущее поколение)

    % lastROUTEant = ROUTEant(1:end,end);

    % локальное обновление феромона, после  каждого муравья

    for tL = 1:n

        xL = ROUTE(tL);

        yL = ROUTE(tL+1);

        % считаем новый феромон

        tao(xL,yL) = (1-p)\*tao(xL,yL) + p\*ph;

        tao(yL,xL) = (1-p)\*tao(yL,xL) + p\*ph;

    end

    end

% --------------------------ГЛОБАЛЬНОЕ ОБНОВЛЕНИЕ--------------------------

    % Испаряем феромоны "старого" пути е - коэффициент испарения

    tao(tao < 2.500000000000000e-150) = 2.500000000000000e-150;

        % для каждого города

        for t = 1:n

            xG = bestROUTE(t);

            yG = bestROUTE(t+1);

            % считаем новый феромон

            tao(xG,yG) = tao(xG,yG) + e\*(Q/bestDIST);

            tao(yG,xG) = tao(yG,xG) + e\*(Q/bestDIST);

        end

end

% строим графику

citiesOP(:,[1,2]) = cities(bestROUTE(:),[1,2]);

plot([citiesOP(:,1);citiesOP(1,1)],[citiesOP(:,2);citiesOP(1,2)],'.r-')

disp (num2str(bestDIST))

msgbox ('Выполнено!')

clearvars -except cities bestDIST bestROUTE iter

toc

Задание 3. Напишите алгоритм, который позволит вам сразиться в крестики - нолики с ПК. Т.е. необходимо реализовать дерево минимакс. По желанию можно произвести в нем альфа и бета отсечения.

#!/usr/bin/env python3

from math import inf as infinity

from random import choice

import platform

import time

from os import system

HUMAN = -1

COMP = +1

board = [

    [0, 0, 0],

    [0, 0, 0],

    [0, 0, 0],

]

def evaluate(state):

    if wins(state, COMP):

        score = +1

    elif wins(state, HUMAN):

        score = -1

    else:

        score = 0

    return score

def wins(state, player):

    win\_state = [

        [state[0][0], state[0][1], state[0][2]],

        [state[1][0], state[1][1], state[1][2]],

        [state[2][0], state[2][1], state[2][2]],

        [state[0][0], state[1][0], state[2][0]],

        [state[0][1], state[1][1], state[2][1]],

        [state[0][2], state[1][2], state[2][2]],

        [state[0][0], state[1][1], state[2][2]],

        [state[2][0], state[1][1], state[0][2]],

    ]

    if [player, player, player] in win\_state:

        return True

    else:

        return False

def game\_over(state):

    return wins(state, HUMAN) or wins(state, COMP)

def empty\_cells(state):

    cells = []

    for x, row in enumerate(state):

        for y, cell in enumerate(row):

            if cell == 0:

                cells.append([x, y])

    return cells

def valid\_move(x, y):

    if [x, y] in empty\_cells(board):

        return True

    else:

        return False

def set\_move(x, y, player):

    if valid\_move(x, y):

        board[x][y] = player

        return True

    else:

        return False

def minimax(state, depth, player):

    if player == COMP:

        best = [-1, -1, -infinity]

    else:

        best = [-1, -1, +infinity]

    if depth == 0 or game\_over(state):

        score = evaluate(state)

        return [-1, -1, score]

    for cell in empty\_cells(state):

        x, y = cell[0], cell[1]

        state[x][y] = player

        score = minimax(state, depth - 1, -player)

        state[x][y] = 0

        score[0], score[1] = x, y

        if player == COMP:

            if score[2] > best[2]:

                best = score  # max value

        else:

            if score[2] < best[2]:

                best = score  # min value

    return best

def clean():

    os\_name = platform.system().lower()

    if 'windows' in os\_name:

        system('cls')

    else:

        system('clear')

def render(state, c\_choice, h\_choice):

    chars = {

        -1: h\_choice,

        +1: c\_choice,

        0: ' '

    }

    str\_line = '---------------'

    print('\n' + str\_line)

    for row in state:

        for cell in row:

            symbol = chars[cell]

            print(f'| {symbol} |', end='')

        print('\n' + str\_line)

def ai\_turn(c\_choice, h\_choice):

    depth = len(empty\_cells(board))

    if depth == 0 or game\_over(board):

        return

    clean()

    print(f'Computer turn [{c\_choice}]')

    render(board, c\_choice, h\_choice)

    if depth == 9:

        x = choice([0, 1, 2])

        y = choice([0, 1, 2])

    else:

        move = minimax(board, depth, COMP)

        x, y = move[0], move[1]

    set\_move(x, y, COMP)

    time.sleep(1)

def human\_turn(c\_choice, h\_choice):

    depth = len(empty\_cells(board))

    if depth == 0 or game\_over(board):

        return

    # Dictionary of valid moves

    move = -1

    moves = {

        1: [0, 0], 2: [0, 1], 3: [0, 2],

        4: [1, 0], 5: [1, 1], 6: [1, 2],

        7: [2, 0], 8: [2, 1], 9: [2, 2],

    }

    clean()

    print(f'Human turn [{h\_choice}]')

    render(board, c\_choice, h\_choice)

    while move < 1 or move > 9:

        try:

            move = int(input('Use numpad (1..9): '))

            coord = moves[move]

            can\_move = set\_move(coord[0], coord[1], HUMAN)

            if not can\_move:

                print('Bad move')

                move = -1

        except (EOFError, KeyboardInterrupt):

            print('Bye')

            exit()

        except (KeyError, ValueError):

            print('Bad choice')

def main():

    clean()

    h\_choice = ''  # X or O

    c\_choice = ''  # X or O

    first = ''  # if human is the first

    # Human chooses X or O to play

    while h\_choice != 'O' and h\_choice != 'X':

        try:

            print('')

            h\_choice = input('Choose X or O\nChosen: ').upper()

        except (EOFError, KeyboardInterrupt):

            print('Bye')

            exit()

        except (KeyError, ValueError):

            print('Bad choice')

    # Setting computer's choice

    if h\_choice == 'X':

        c\_choice = 'O'

    else:

        c\_choice = 'X'

    # Human may starts first

    clean()

    while first != 'Y' and first != 'N':

        try:

            first = input('First to start?[y/n]: ').upper()

        except (EOFError, KeyboardInterrupt):

            print('Bye')

            exit()

        except (KeyError, ValueError):

            print('Bad choice')

    # Main loop of this game

    while len(empty\_cells(board)) > 0 and not game\_over(board):

        if first == 'N':

            ai\_turn(c\_choice, h\_choice)

            first = ''

        human\_turn(c\_choice, h\_choice)

        ai\_turn(c\_choice, h\_choice)

    # Game over message

    if wins(board, HUMAN):

        clean()

        print(f'Human turn [{h\_choice}]')

        render(board, c\_choice, h\_choice)

        print('YOU WIN!')

    elif wins(board, COMP):

        clean()

        print(f'Computer turn [{c\_choice}]')

        render(board, c\_choice, h\_choice)

        print('YOU LOSE!')

    else:

        clean()

        render(board, c\_choice, h\_choice)

        print('DRAW!')

    exit()

if \_\_name\_\_ == '\_\_main\_\_':

    main()