Министерство образования и науки Российской Федерации Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Севастопольский государственный университет»

ОТЧЕТ

о выполнении лабораторной работы № 3 по дисциплине

«Методы и системы искусственного интеллекта»

Выполнил:

ст. гр. ИС/б-22-1-о

Гюнтер М. Ю .

Проверил:

доцент кафедры

“Информационные системы”

Бондарев В. Н.

Севастополь,2025

# Цель работы: Исследование информированных методов поиска решений задач в пространстве состояний, приобретение навыков программирования интеллектуальных агентов, планирующих действия на основе методов эвристического поиска решений задач.

# Ход работы

## Алгоритм A\*

Для метода aStarSearch() среды AI Pacman с помощью эвристической функции манхэттенского расстояния, которая на координатной плоскости с возможностью перемещения в 4 направлениях отвечает требованиям полноты и монотонности, был реализован алгоритм A\* (листинг ‎2.1).

Листинг ‎2.1 – Алгоритм A\*

def aStarSearch(problem, heuristic=nullHeuristic):

"""

Находит узел с наименьшей комбинированной стоимостью, включающей эвристику

"""

"\*\*\* ВСТАВЬТЕ ВАШ КОД СЮДА \*\*\*"

OPEN = util.PriorityQueue()

CLOSED = set({})

TOTAL\_COSTS\_AND\_ESTS = {}

def pushSuccessors(parentState, OPEN, CLOSED, TOTAL\_COSTS\_AND\_ESTS):

parentTotalCost = TOTAL\_COSTS\_AND\_ESTS[parentState][0]

children = problem.getSuccessors(parentState)

for childState, childAct, parentChildCost in children:

childCost = parentTotalCost + parentChildCost

childEstCost = childCost + heuristic(childState, problem)

if childEstCost < TOTAL\_COSTS\_AND\_ESTS.get(childState, (float('inf'), float('inf'), None, None))[1]:

CLOSED.discard(childState)

TOTAL\_COSTS\_AND\_ESTS[childState] = (childCost, childEstCost, childAct, parentState)

OPEN.update(childState, childEstCost)

def findPath(goalState, TOTAL\_COSTS\_AND\_ESTS):

goalAct = TOTAL\_COSTS\_AND\_ESTS[goalState][2]

parentState = TOTAL\_COSTS\_AND\_ESTS[goalState][3]

parent = TOTAL\_COSTS\_AND\_ESTS[parentState]

path = [goalAct]

while parent[3]:

parentAct = parent[2]

parentState = parent[3]

parent = TOTAL\_COSTS\_AND\_ESTS[parentState]

path.append(parentAct)

path.reverse()

return path

startState = problem.getStartState()

TOTAL\_COSTS\_AND\_ESTS[startState] = (0, 0, None, None)

OPEN.push(startState, 0)

while not OPEN.isEmpty():

state = OPEN.pop()

if problem.isGoalState(state):

return findPath(state, TOTAL\_COSTS\_AND\_ESTS)

CLOSED.add(state)

pushSuccessors(state, OPEN, CLOSED, TOTAL\_COSTS\_AND\_ESTS)

Результаты работы алгоритма A\* с эвристической функцией манхэттенского расстояния, для которой на координатном пространстве с возможностью перемещения в 4 направлениях выполняются условия как допустимости, так и монотонности, для лабиринта bigMaze приведены на рисунке ‎2.1.

Изображение выглядит как текст, Шрифт, алгебра, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок ‎2.1 – Результаты работы алгоритма A\* с эвристической функцией манхэттенского расстояния

В результате обследования Pacman-ом лабиринта им было раскрыто 549 вершин, что соответствует указанному в методических указаниях числу вершин. При этом полученный результат правда несколько меньше по сравнению с полученным ранее для алгоритма равных цен, в ходе работы которого было раскрыто 620 вершин.

Приведем также схему, отображающую порядок обследования Pacman-ом вершин лабиринта, в которой яркость окраски вершины уменьшается в зависимости от промежутка времени, на котором Pacman обследовал вершину (рисунок ‎2.2).

Изображение выглядит как шаблон, искусство, Красочность, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок ‎2.2 – Схема обследования Pacman-ом вершин лабиринта при поиске алгоритмом A\* с эвристической функцией манхэттенского расстояния

На схеме хорошо видно, как из-за движения в неоптимальном с точки зрения манхэттенского расстояния направлении Pacman сначала обследует контуры равных цен в правой части лабиринта. Тем не менее, когда Pacman выходит на участок лабиринта, при движении по которому эвристическая функция убывает, он не обследует правый участок лабиринта, и вместо этого движется напрямую к целевой вершине.

Результаты работы автооценивателя autograder.py для алгоритма A\* с эвристической функцией манхэттенского расстояния приведены на рисунке ‎2.3.

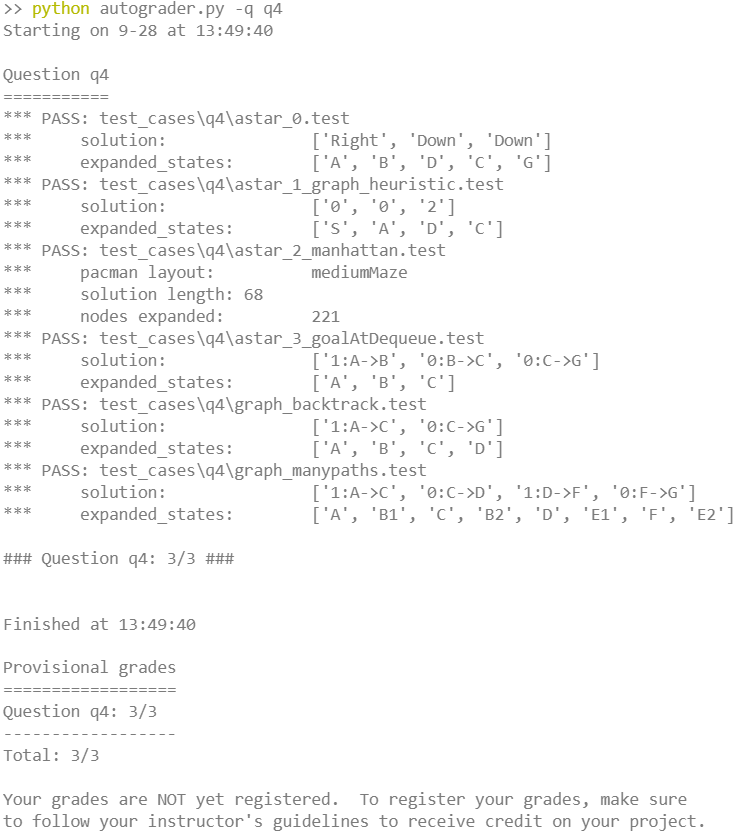


Рисунок ‎2.3 – Результаты работы автооценивателя для алгоритма A\* с эвристической функцией манхэттенского расстояния

Для алгоритма A\* временная и пространственная сложности являются экспоненциальными.

## Поиск всех углов

Для класса CornersProblem среды AI Pacman были определены конструктор \_\_init\_\_(), а также методы getStartState(), isGoalState() и getSuccessors() (листинг ‎2.2).

Листинг ‎2.2 – Класс CornersProblem

class CornersProblem(search.SearchProblem):

"""

Эта задача поиска находит пути через все четыре угла схемы игры.

Вы должны выбрать подходящее пространство состояний и функцию-преемник.

"""

def \_\_init\_\_(self, startingGameState):

"""

Хранит стены, исходную позицию Пакмана и углы.

"""

self.walls = startingGameState.getWalls()

self.startingPosition = startingGameState.getPacmanPosition()

top, right = self.walls.height-2, self.walls.width-2

self.corners = ((1,1), (1,top), (right, 1), (right, top))

for corner in self.corners:

if not startingGameState.hasFood(\*corner):

print('Warning: no food in corner ' + str(corner))

self.\_expanded = 0 # НЕ МЕНЯЙТЕ; Количество раскрытых поисковых узлов

# Пожалуйста, добавьте сюда любой код, который вы хотели бы использовать

# при инициализации задачи

"\*\*\* ВСТАВЬТЕ ВАШ КОД СЮДА \*\*\*"

self.startingGameState = startingGameState

def getStartState(self):

"""

Возвращает начальное состояние (в вашем пространстве состояний, а

не в полном пространстве состояний игры Pacman)

"""

"\*\*\* ВСТАВЬТЕ ВАШ КОД СЮДА \*\*\*"

return (self.startingPosition, ())

def isGoalState(self, state):

"""

Проверяет, является ли это состояние поиска целевым состоянием задачи.

"""

"\*\*\* ВСТАВЬТЕ ВАШ КОД СЮДА \*\*\*"

stateCorners = state[1]

return len(stateCorners) == 4

def getSuccessors(self, state):

"""

Возвращает состояния-преемники, действия, и стоимость 1.

Как отмечено в search.py:

Для данного состояния возвращает список из триплетов (successor,

action, stepCost), где 'successor' - это преемник текущего состояния,

'action' - это действие, необходимое для его достижения,

'stepCost' - затраты для шага перхода к этому преемнику.

"""

stateCrds, stateCorners = state

successors = []

for action in [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST]:

# Добавьте состояние-приемник в список приемников, если действие является

# допустимым

# Ниже фрагмент кода, который выясняет, не попадает ли новая позиция на

# стену лабиринта:

# x,y = currentPosition

# dx, dy = Actions.directionToVector(action)

# nextx, nexty = int(x + dx), int(y + dy)

# hitsWall = self.walls[nextx][nexty]

"\*\*\* ВСТАВЬТЕ ВАШ КОД СЮДА \*\*\*"

x, y = stateCrds

dx, dy = Actions.directionToVector(action)

nextX, nextY = int(x + dx), int(y + dy)

hitsWalls = self.walls[nextX][nextY]

if not hitsWalls:

nextStateCrds = (nextX, nextY)

nextStateCorners = stateCorners

if (nextX, nextY) in self.corners and (nextX, nextY) not in stateCorners:

# Create a new independent tuple based on the stateCorners tuple

nextStateCorners = stateCorners + (nextStateCrds, )

successors.append(((nextStateCrds, nextStateCorners), action, 1))

self.\_expanded += 1 # НЕ МЕНЯЙТЕ!

return successors

def getCostOfActions(self, actions):

"""

Возвращает стоимость определенной последовательности действий. Если эти

действия включают недопустимый ход, вщзвращает 999999.

"""

if actions == None: return 999999

x,y= self.startingPosition

for action in actions:

dx, dy = Actions.directionToVector(action)

x, y = int(x + dx), int(y + dy)

if self.walls[x][y]: return 999999

return len(actions)

В качестве состояния, используемого для задания пространства состояний, было решено использовать кортеж ((x, y), (c1, c2, … cn)), состоящий из кортежа координат местоположения агента и кортежа, содержащего кортежи координат всех посещенных им углов.

Для метода breadthFirstSearch() среды AI Pacman был модифицирован алгоритм поиска в ширину для учета измененного состояния пространства состояний (листинг ‎2.3).

Листинг ‎2.3 – Алгоритм поиска в ширину

OPEN = util.Queue()

CLOSED = set({})

PATHS = {}

def pushSuccessors(parentState, OPEN, CLOSED, PATHS):

parentPath = PATHS[parentState]

children = problem.getSuccessors(parentState)

for childState, childAct, \_ in children:

if childState not in CLOSED and childState not in PATHS:

childPath = parentPath + [childAct]

PATHS[childState] = childPath

OPEN.push(childState)

startState = problem.getStartState()

PATHS[startState] = []

OPEN.push(startState)

while not OPEN.isEmpty():

state = OPEN.pop()

if problem.isGoalState(state):

return PATHS[state]

CLOSED.add(state)

pushSuccessors(state, OPEN, CLOSED, PATHS)

Отметим, что по сравнению с предыдущим подходом, основанном на состоянии, представленном в виде кортежа координат (x, y), теперь хранение данных как в очереди OPEN, так и в множестве CLOSED, осуществляется не на основании кортежа координат, а на основании полного состояния, включающего как кортеж координат (x, y), так и кортеж посещенных углов (c1, c2, … cn). Таким образом одна и та же вершина может неоднократно включаться в оба списка в зависимости от того, какие углы были пройдены агентом.

Фактически в ходе поиска агент сначала обнаружит оптимальный путь из своей начальной позиции к одному из углов, после чего он найдет оптимальный путь из этого угла к другому углу, и так далее, пока не будет получено целевое состояние, включающее в себя все 4 угла. Такой подход позволяет найти наикратчайший путь из стартовой вершины, проходящий через все 4 угла лабиринта.

Результаты работы алгоритма BFS для двух лабиринтов разного размера для проблемы поиска всех углов приведены на рисунке ‎2.4.

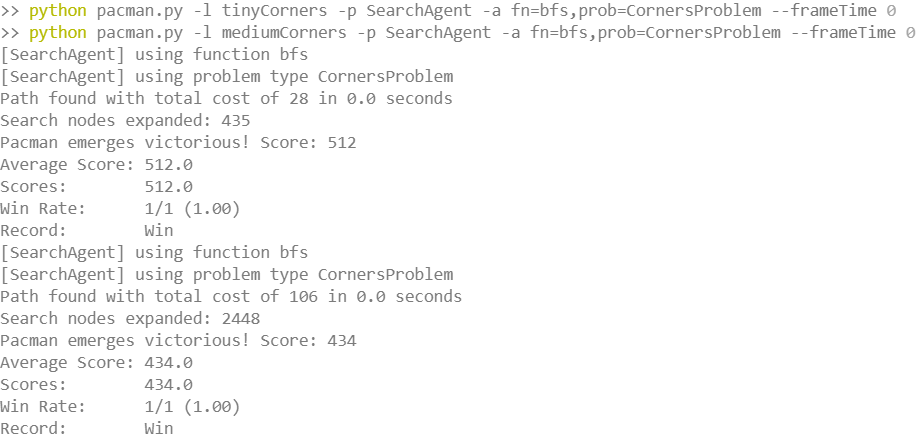


Рисунок ‎2.4 – Результаты работы алгоритма BFS для двух лабиринтов разного размера для проблемы поиска всех углов

По результатам работы алгоритма BFS, решающего задачу поиска всех углов в лабиринте tinyMaze, можно говорить о корректной реализации проблемы CornersProblem и алгоритма breadthFirstSearch() – количество раскрытых вершин, равное 28, совпадает с указанным в методических указаниях. Для лабиринта mediumMaze также получен верный результат – число раскрытых в процессе поиска вершин превышает 2000. Очевидно, что для задачи небольшой размерности такое число раскрытий достаточно велико, и необходимо прибегать к использованию эвристической функции для ускорения процесса поиска.

Результаты работы автооценивателя autograder.py для задачи поиска всех углов приведены на рисунке ‎2.5.

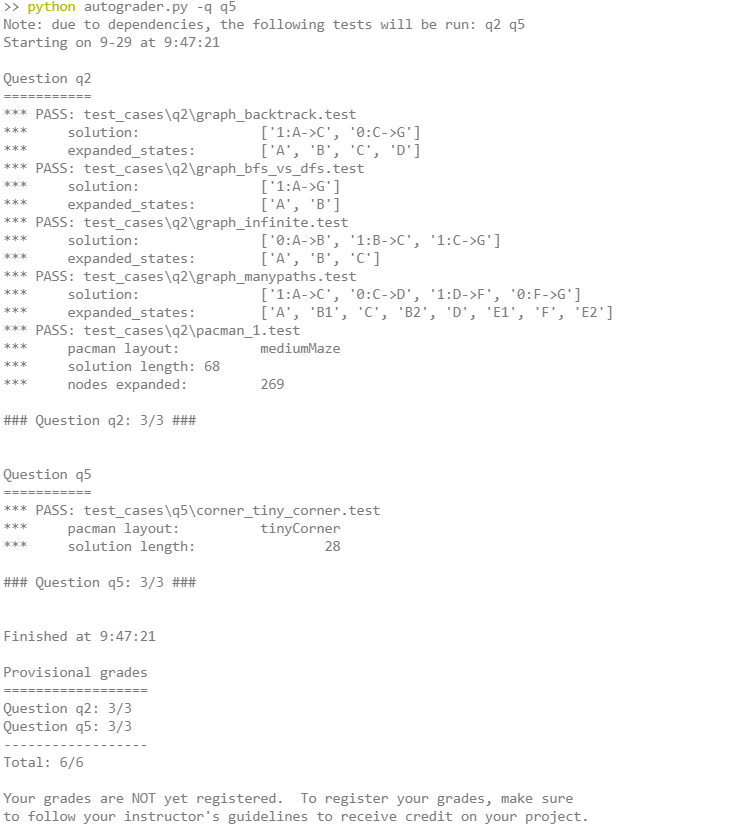


Рисунок ‎2.5 – Результаты работы автооценивателя для задачи поиска всех углов

## Определение эвристической функции

Для метода uniformCostSearch() среды AI Pacman был реализован алгоритм поиска в ширину (листинг ‎2.4).

Листинг ‎2.4 – Алгоритм равных цен

def uniformCostSearch(problem):

"""Находит узел минимальной стоимости """

"\*\*\* ВСТАВЬТЕ ВАШ КОД СЮДА \*\*\*"

def pushSuccessors(parentCrds, OPEN, CLOSED, PRIORITIES):

parentCost = PRIORITIES[parentCrds][0]

children = problem.getSuccessors(parentCrds)

for childCrds, childAct, parentChildCost in children:

if childCrds not in CLOSED:

childCost = parentCost + parentChildCost

OPEN.update(childCrds, childCost)

if childCost < PRIORITIES.get(childCrds, (float('inf'), None, None))[0]:

PRIORITIES[childCrds] = (childCost, childAct, parentCrds)

def findPath(goalCrds, PRIORITIES):

goalAct = PRIORITIES[goalCrds][1]

parentCrds = PRIORITIES[goalCrds][2]

parent = PRIORITIES[parentCrds]

path = [goalAct]

while parent[2] != None:

parentAct = parent[1]

path.append(parentAct)

parent = PRIORITIES[parent[2]]

path.reverse()

return path

OPEN = util.PriorityQueue()

CLOSED = set({})

PRIORITIES = {}

startStateCrds = problem.getStartState()

PRIORITIES[startStateCrds] = (0, None, None)

OPEN.push(startStateCrds, 0)

while not OPEN.isEmpty():

stateCrds = OPEN.pop()

if problem.isGoalState(stateCrds):

return findPath(stateCrds, PRIORITIES)

CLOSED.add(stateCrds)

pushSuccessors(stateCrds, OPEN, CLOSED, PRIORITIES)

Результаты работы алгоритма UCS для трех лабиринтов с существенными различиями в стоимостях путей приведены на рисунке ‎2.7.

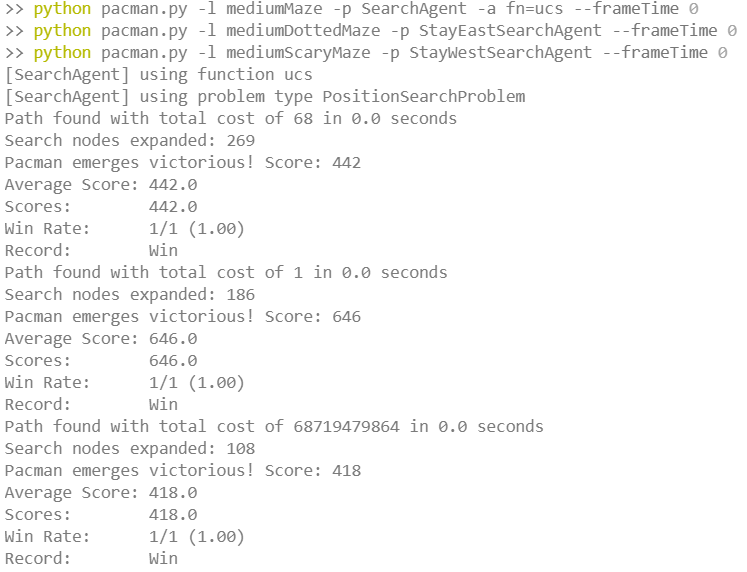


Рисунок ‎2.7 – Результаты работы алгоритма UCS для трех лабиринтов с существенными различиями в стоимостях путей

Результаты работы автооценивателя autograder.py для алгоритма UCS приведены на рисунке ‎2.8.

Изображение выглядит как текст, снимок экрана, письмо, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок ‎2.8 – Результаты работы автооценивателя для алгоритма UCS

Для алгоритма равных цен при поиске по дереву временная и пространственная сложности являются экспоненциальными, и пропорциональны , где B – коэффициент ветвления, С – стоимость оптимального решения, c – средняя минимальная стоимость действия.

# Вывод:

В результате выполнения лабораторной работы были изучены алгоритмы “слепого”, или неинформированного поиска – поиск в ширину (Breadth First Search, BDF), поиск в глубину (Depth First Search, DFS), алгоритм равных цен (Uniform Cost Search, UCS). Были детально рассмотрены отличия между алгоритмами, как по критериям оценки эффективности алгоритмов, таких, как полнота, оптимальность, минимальность, временная сложность, пространственная сложность, так и по общим принципам работы алгоритмов. Каждый из исследованных алгоритмов слепого поиска был реализован в среде AI Pacman с использованием специальных структур данных – очереди, стека и очереди с приоритетами, необходимых для реализации обозначенных алгоритмов.