دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

گزارش دوم

پیاده سازی هیوریستیک برای حل مساله ۸ پازل

نگارش

سروش آریانا

استاد

دکتر مهدی قطعی

اسفند ۹۹

مقدمه

مساله ۸ پازل یکی از مسائل معروف در هوش مصنوعی برای پیاده سازی با روش های مختلف جستجو و الگوریتم های متفاوت است. در این گزارش به حل این مساله با الگوریتم جستجو A* و تابع هیوریستیکی که در متن به توضیح آن خواهیم پراخت، می‌پردازیم. سورس کد برنامه در بخش پیوست ها آورده شده است.

۱- ساختار برنامه

در ابتدای برنامه حالت نهایی و حالت ابتدایی پازل مشخص شده اند و برای آزمایش ورودی های متفاوت می‌توانیم اعداد خانه های حالت ابتدایی را در اول برنامه تغییر دهیم(به جای خانه خالی عدد صفر وارد می‌شود.). بر اساس ورودی داده شده، برنامه ابتدا قابل حل بودن پازل با این ورودی ها را توسط تابع `inversion`، که در بخش های بعدی آن را توضیح داده ایم، چک می‌کند و اگر قابل حل نباشد با نمایش پیغامی غیر قابل حل بودن آن را بیان می‌کند و در غیر این صورت به حل پازل می‌پردازد و برای این کار از الگوریتم `A*` و تابع هیورستیکی به نام `calculate_h`، که در بخش های بعدی آن را شرح می‌دهیم، استفاده می‌کند.در هنگام حل، گره هایی که برای رسیدن به حالت نهایی بررسی شده اند چاپ می‌شوند و در نهایت تعداد کل گره های تولید شده، تعداد گره های بررسی شده، و زمان حل پازل نمایش داده می‌شوند.

۲- تابع INVERSION

در مساله ۸ پازل، بعضی حالت های ورودی غیر قابل حل هستند.برای تشخیص قابل حل بودن، عددی به عنوان شاخص محاسبه می‌شود که اگر این عدد فرد باشد پازل غیر قابل حل، و اگر زوج باشد قابل حل است. در ادامه با مثال نحوه این محاسبه را توضیح خواهیم داد.

۵	۲	۸
۴	۱	۷
	۳	۶

شکل ۱-۲: نمونه حالت قابل حل

ورودی های این پازل را به صورت ۵۲۸۴۱۷۳۶ در نظر می‌گیریم(خانه خالی را نادیده می‌گیریم.). برای هر خانه، تعداد خانه هایی که بعد از آن در جدول ظاهر شده اند و مقدار آن ها از مقدار خانه مورد بررسی ما کمتر است را محاسبه می‌کنیم و در نهایت این تعداد را با هم جمع میزنیم. به عنوان مثال در شکل ۱-۲، پس از بررسی و محاسبه برای هر خانه اعداد به صورت زیر به دست می‌آیند.

برای عدد ۵: چهار خانه (۱،۲،۳،۴)

برای عدد ۲: یک خانه (۱)

برای عدد ۸: پنج خانه (۱،۳،۴،۶،۷)

برای عدد ۴: دو خانه (۱،۳)

برای عدد ۱: صفر خانه

برای عدد ۷: دو خانه (۳،۴)

برای عدد ۳: صفر خانه

برای عدد ۶: صفر خانه
پس خواهیم داشت ۱۴=۰+۰+۲+۰+۲+۵+۱+۴ و عدد شاخص برابر با ۱۴ خواهد بود که عددی زوج است و در نتیجه حالت مذکور، قابل حل است.
در زیر نمونه ای از حالت غیرقابل حل ذکر شده است.

۱	۲	۳
۴	۵	۶
	۸	۷

شکل ۲-۲: نمونه حالت غیر قابل حل

با بررسی شکل ۲-۲ به همان روش قبل، می‌بینیم که عدد شاخص برابر با ۱ خواهد شد که عددی فرد است و در نتیجه این حالت غیر قابل حل خواهد بود.

۳- تابع هیوریستیک

این تابع که `calculate_h` نام دارد، به این صورت عمل می‌کند که تعداد خانه هایی که در حال حاضر از سطر حالت نهایی خود خارج هستند و تعداد خانه هایی که در حال حاضر از ستون حالت نهایی خود خارج هستند را محاسبه می‌کند و مجموع آن ها را برمی‌گرداند. در ادامه با ارائه مثالی به توضیح نحوه عملکرد این تابع می‌پردازیم.

مثلا در شکل ۱-۲، اعداد ۱، ۳، ۵، ۶، ۷، و ۸ (شش عدد) از سطر حالت نهایی خود و اعداد ۱، ۳، ۵، ۷، و ۸ (پنج عدد) از ستون حالت نهایی خود خارج هستند؛ پس داریم ۱۱=۶+۵ و تابع هیوریستیک ما مقدار ۱۱ را برمی‌گرداند. برنامه بر اساس این تابع و البته بر اساس الگوریتم `A*` در هر مرحله سعی می‌کند حالتی که نسبت به بقیه حالات ممکن مقدار هیوریستیک کمتری دارد را انتخاب کند و پیش برود تا به مقدار ۰ که مربوط به حالت نهایی پازل است، برسد.

۴- مقایسه ورودی های مختلف

در جدول زیر عملکرد برنامه با ورودی های مختلف مقایسه شده است. همان طور که گفته شد، در پایان برنامه تعداد گره هایی که برای رسیدن به جواب بررسی شده اند(که آن را `step` نام گذاری می‌کنیم)، تعداد کل گره های تولید شده و زمان حل پازل چاپ می‌شوند که مقایسه ما هم بر همین اساس خواهد بود.

ورودی پازل	step	تعداد کل گره ها	زمان حل(ثانیه)
۱۲۵۴۰۸۳۶۷	۳۸۸	۶۶۷	۴,۶۰۹۱۳۰۱۹۴
۸۷۶۴۰۳۵۲۱	۳۴۱	۵۶۸	۳,۷۰۲۵۴۳۷۰۱
۱۴۲۶۳۵۷۸۰	۹۰	۱۵۴	۰,۲۵۷۹۱۸۱۴۹
۳۲۷۶۰۱۸۴۵	۵۳۶	۹۰۹	۸,۴۰۳۸۳۰۸۸۷
۸۷۶۵۴۳۲۱۰	۵۰۳	۸۶۰	۷,۸۲۸۲۰۳۷۸۴
۳۴۷۵۰۸۶۲۱	۳۲۵	۵۵۸	۳,۲۳۶۹۱۴۶۶۸
۶۱۷۸۰۵۲۳۴	۵۲۶	۹۰۱	۸,۲۷۷۰۳۰۳۴۸
۷۶۰۸۴۳۵۲۱	۵۵۷	۹۲۹	۸,۸۹۳۲۰۰۲۵۸
۱۰۲۶۴۳۷۸۵	۸۶	۱۴۶	۰,۲۲۱۸۸۴۲۶۶

جدول ۱-۴: مقایسه ورودی های مختلف

منابع و مراجع

https://math.stackexchange.com/

https://www.geeksforgeeks.org/

فایل pdf به نشانی:

https://cse.iitk.ac.in/users/cs365/2009/ppt/13jan_Aman.pdf

سلیمانی، بهزاد و یاوری، مجید و شهابیان، اسماعیل و نیکدست، مهدی،۱۳۸۷،پازل 8تایی و یک تابع هیوریستیک مناسب برای حل این مسئله به روشIDA ،یازدهمین کنفرانس دانشجویی مهندسی برق ایران،زنجان،،، https://civilica.com/doc/48746

پیوست‌ها

سورس کد برنامه به زبان پایتون

```
import copy
import math
import timeit
import sys

import numpy as np

final_state = [[1, 2, 3],
               [4, 5, 6],
               [7, 8, 0]]
initial_state = [[1, 0, 2],
                 [6, 4, 3],
                 [7, 8, 5]]

len_row = len(final_state[0])  # number of items in each row
len_column = len(final_state)  # number of items in each column
space = []  # location of zero
open_set = [initial_state]
all_h = []
close_set = [initial_state]


def calculate_h(matrix):  # my heuristic
    h = 0
    if matrix is None:
        return math.inf
    for i in range(len_row):
        for j in range(len_column):
            exist_row = False
            exist_column = False
            if final_state[i][j] == 0:
                continue  # don't use 0 in calculating h
            for d in range(len_column):  # search in column
                if final_state[i][j] == matrix[i][d]:
                    exist_row = True
            if not exist_row:
                h = h + 1
            for k in range(len_row):  # search in row
                if final_state[i][j] == matrix[k][j]:
                    exist_column = True
            if not exist_column:
                h = h + 1
    return h


def find_location(number, matrix):
    return [[index, row.index(number)] for index, row in
enumerate(matrix) if number in row][0]


def inversions(s):
    flat_list = []
    for sublist in s:
```

```
        for item in sublist:
            flat_list.append(item)
    flat_list = np.array(flat_list)
    k = flat_list[flat_list != 0]
    tinv = 0
    for i in range(len(k) - 1):
        b = np.array(np.where(k[i + 1:] < k[i])).reshape(-1)
        tinv += len(b)
    return tinv


def check(matrix):
    for d in range(len(close_set)):
        same_matrix = True
        for i in range(len_row):
            for j in range(len_column):
                if close_set[d][i][j] != matrix[i][j]:
                    same_matrix = False
        if same_matrix:
            return False
    close_set.append(matrix)
    return True


def move_up():
    if space[0] == 0:
        return None  # cant move up
    row = space[0]  # row of zero
    column = space[1]  # column of zero
    new_matrix = copy.deepcopy(initial_state)
    new_matrix[row - 1][column] = initial_state[row][column]  # swap
with zero
    new_matrix[row][column] = initial_state[row - 1][column]  # swap
with zero
    return new_matrix


def move_down():
    if space[0] == len_column - 1:
        return None  # cant move down
    row = space[0]  # row_of_zero
    column = space[1]  # column_of_zero
    new_matrix = copy.deepcopy(initial_state)
    new_matrix[row + 1][column] = initial_state[row][column]  # swap
with zero
    new_matrix[row][column] = initial_state[row + 1][column]  # swap
with zero
    return new_matrix


def move_right():
    if space[1] == len_row - 1:
        return None  # cant move right
    row = space[0]  # row_of_zero
    column = space[1]  # column_of_zero
    new_matrix = copy.deepcopy(initial_state)
    new_matrix[row][column + 1] = initial_state[row][column]  # swap
with zero
```

```
    new_matrix[row][column] = initial_state[row][column + 1]  # swap
with zero
    return new_matrix


def move_lef():
    if space[1] == 0:
        return None  # cant move left
    row = space[0]  # row_of_zero
    column = space[1]  # column_of_zero
    new_matrix = copy.deepcopy(initial_state)
    new_matrix[row][column - 1] = initial_state[row][column]  # swap
with zero
    new_matrix[row][column] = initial_state[row][column - 1]  # swap
with zero
    return new_matrix


if __name__ == '__main__':
    # Count inversions in given 8 puzzle
    invCount = inversions(initial_state)
    # return true if inversion count is even.
    if invCount % 2 != 0:
        print('Not Solvable')
        sys.exit()
    print_format = '{:' + str(len_row) + '}'
    start = timeit.default_timer()
    all_h.append(calculate_h(initial_state))
    step = 0
    number_of_nodes = 1
    while len(open_set) > 0:
        step = step + 1
        min_h_index = all_h.index(min(all_h))
        initial_state = open_set.pop(min_h_index)
        current_h = all_h.pop(min_h_index)
        print('\n'.join([''.join([print_format.format(item) for item
in row])
                         for row in initial_state]))
        print()
        if current_h == 0:
            break
        space = find_location(0, initial_state)
        move = [move_right(), move_down(), move_lef(), move_up()]
        for matrix in move:
            if matrix is not None and check(matrix):
                open_set.append(matrix)
                all_h.append(calculate_h(matrix))
                number_of_nodes = number_of_nodes + 1
    stop = timeit.default_timer()
    print("number of nodes: " + str(number_of_nodes))
    print("step: " + str(step))
    print('Time: ', stop - start)
```