# COMP3211:
# Fundamentals of Artificial Intelligence

**Homework Assignment 2**

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Release Date: Oct. 17, 2024

## The HKUST Academic Honor Code

Honesty and integrity are central to the academic work of HKUST. Students of the University must observe and uphold the highest standards of academic integrity and honesty in all the work they do throughout their program of study. As members of the University community, students have the responsibility to help maintain the academic reputation of HKUST in its academic endeavors.

## 1 Introduction

In this homework, we will have two problems. First, we will test the performance of different heuristics in A* search for the 8-puzzle. Then, we will solve a cryptarithmetic problem by hand. The submission deadline for this homework is **11:59 pm, Nov. 6, 2024**. Please start as early as possible, and feel free to discuss with Shuling Zhao (szhaoax@connect.ust.hk) for problem 1 and Jiabo Ma (jmabq@connect.ust.hk) for problem 2 if you have any questions.

This assignment should be solved individually. No collaboration, sharing of solutions, or exchange of models is allowed. Please, do not directly copy existing code from anywhere other than your previous solutions, or the previous master solution. We will check assignments for duplicates. See below for more details about the homework.

## 2 Problem 1: A* Search for the 8-puzzle (60 points)

In this part, your task is to write a program to solve the 8-puzzle with A* search and compare the performance of different heuristics.

Given a 3 × 3 board (containing 9 squares) with 8 tiles (each numbered from 1 to 8) and 1 empty space, the objective of the 8-puzzle is to place the numbers to match the final configuration (the goal state) using the empty space. We can slide four adjacent tiles (left, right, above, and below) into the empty space.

The cost of one move is 1. We represent the empty space with “0”. We define the goal state as follows:

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 8 | 0 | 4 |
| 7 | 6 | 5 |

The arguments of your program are:

- `-p puzzle`
- `-H heuristic`
- `-o output file`

For how to set arguments in Python, you can refer to https://docs.python.org/3/how-to/argparse.html.

**Puzzle:** The input file of your program which contains a $3 \times 3$ matrix representing the initial configuration of the tiles and the empty space, which is the initial state of the 8-puzzle. An example is shown below:

$$\begin{matrix} 1 & 2 & 3 \\ 7 & 8 & 4 \\ 6 & 5 & 0 \end{matrix}$$

**Heuristic:** You should implement the **ADMISSIBLE** heuristics as described in Sec. 2.1. Use the argument to specify which heuristic your A* search uses.

**Output file:** Your program should output the solution to the 8-puzzle in the output file. The solution is a path consisting of all the consecutive states from the initial state to the goal state. The first line of the file should be your student ID. The following is an example with the Manhattan distance as the heuristic function:

```
(output file)
10123456
1 2 3
7 8 4
6 5 0

1 2 3
7 8 4
6 0 5

1 2 3
7 8 4
0 6 5

1 2 3
0 8 4
7 6 5

1 2 3
8 0 4
7 6 5
```

In this problem, we provide puzzle_1.txt, puzzle_2.txt, puzzle_3.txt, puzzle_4.txt and puzzle_5.txt as the input files each containing an initial state and problem1_skeleton_code.py containing the skeleton code for you. Feel free to modify the code in problem1_skeleton_code.py to facilitate your implementation as long as the resulting program accepts the arguments above and returns the required results. You **CANNOT USE** any advanced packages (e.g., astar) for your implementation of A* search. Please show the required results in your PDF and provide the source code, as well as a document (e.g., in markdown format) to show how to run your code.

## 2.1 Implement A* Search (40 points)

Consider the following four heuristic functions:

1) Misplaced tiles: The number of misplaced tiles.

2) Manhattan distance: The sum of the absolute differences between the current and target positions of each tile.

3) Nilsson (1971): $h(n) = P(n) + 3S(n)$ where $P(n)$ is the Manhattan distance and $S(n)$ is a sequence score obtained by checking around the non-central squares in turn, allotting 2 for every tile not followed by its proper successor (the square clockwise to it) and 0 for every other tile, except that a piece in the center scores 1.

   We give an example to illustrate the computation of the sequence score. Given the following state, we first check the tiles on the non-central squares. The successor of tile 1 is tile 2, which is the same as its successor in the goal state, so we allot 0 for tile 1. The successor of tile 2 is tile 6, which is different from its successor in the goal state (tile 3), so we allot 2 for tile 2. After checking every tile on the non-central squares, we check if there is a tile in the center. We find tile 7 in the center, so we allot 1 for tile 7. The sum of all the allotted scores is the sequence score.

```
1 2 6
8 7 4
3 5 0
```

4) Mostow and Prieditis (1989): The number of tiles out of row + the number of tiles out of column.

   We illustrate the heuristic with the following state. Tile 3 is out of row, for its current position is in the third row, and its target position is in the first row. Similarly, tiles 6 and 7 are out of row. The number of tiles out of row is 3. Tile 3 is out of column, for its current position is in the first column, and its target position is in the third column. Similarly, tiles 5, 6 and 7 are out of column. The number of tiles out of column is 4. Therefore, the heuristic value for this state is 3 + 4 = 7.

```
1 2 6
8 7 4
3 5 0
```

**Note that the tiles in the heuristics above only refer to the numbers from 1 to 8 in the matrix. 0 represents the empty space, not a tile.**

### 2.1.1 Identify Heuristic (10 points)

Determine which heuristic is **NOT** admissible and provide a proof using an example.

### 2.1.2 Implement A* Search with Admissible Heuristics (30 points)

Implement your A* search using the **ADMISSIBLE** heuristics.

We provide skeleton code in problem1_skeleton_code.py. You can modify this code to support your implementation. For instance, while the code includes space for all four heuristics, you do not need to implement the inadmissible one. Additionally, You may adjust the allowable values for the argument `heuristic` to include only those representing admissible heuristics.

Please organize your final program submission in a file named YourStudentID_problem1.py. Ensure that the file can accept the required arguments and produce the necessary output. Furthermore, please submit a document (e.g., in markdown format) that includes any steps or dependencies needed to run your code successfully.

## 2.2 Performance Comparison (20 points)

The last two heuristics in Sec. 2.1 purport to be better than either the Manhattan distance or the misplaced tiles. Test these claims by comparing the performance of all four heuristics in Sec. 2.1 from the following two perspectives:

1. **Admissibility** of the four heuristics.
2. **Efficiency** of the resulting algorithms that use the admissible heuristics from Sec. 2.1.2.

Hint: To evaluate admissibility, determine whether each heuristic is admissible. To evaluate efficiency, use the 5 initial states provided in puzzle_*.txt for the 8-puzzle, and compare the number of nodes expanded by the resulting algorithms that use the admissible heuristics from Sec. 2.1.2. Please present your results, analyze them, and provide your conclusions. There is no need to evaluate the efficiency of the inadmissible heuristic.

## 3 Problem 2: Constraint Satisfaction Problem (40 points)

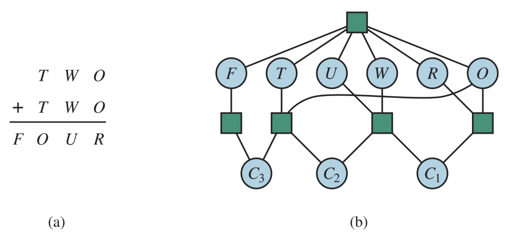

Figure 1: (a) A cryptarithmetic problem. **Each letter stands for a distinct digit**; the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, with the added restriction that **no leading zeroes are allowed**. (b) The constraint hypergraph for the cryptarithmetic problem, showing the *Alldiff* constraint (square box at the top) as well as the column addition constraints (four square boxes in the middle). The variables $C_1$, $C_2$, and $C_3$ represent the carry digits for the three columns from right to left. An *Alldiff* constraint is a global constraint which says that all of the variables involved in the constraint must have different values.

In this part, you need to solve the cryptarithmetic problem shown in Figure 1 by hand. You need to solve the cryptarithmetic problem using the strategy of backtracking with forward checking and minimum-remaining values (MRV) and least-constraining-value (LCV) heuristics. Specifically:

- Specify the variable to be chosen and the assigned value
- List the updated constraints after you assign the value to the chosen variable
- List the remaining legal values for unassigned variables

**Please finish following steps to solve this problem:**
**Step 1**: List all constraints. (15 points)

**Step 2**: Start the backtracking search algorithm with forward checking. (25 points)
(a) Initial legal values.

| F | T | U | W | R | O |
|---|---|---|---|---|---|
| {1} | {2, 3, ..., 9} | {0, 2, 3, ..., 9} | {0, 2, 3, ..., 9} | {0, 2, 3, ..., 9} | {0, 2, 3, ..., 9} |

Please continue backtracking with forward checking to solve this problem. You need to list all the constraints after assigning a value to a variable. Note that if there is a tie, always choose the **smallest number**.

## 4 Submission

Please submit a PDF file including the answer for problems 1 and 2, a Python file and a file to show how to run your code for problem 1 to show your work. For problem 1, the PDF only needs to write your answer for Sec. 2.1.1 and Sec. 2.2. Name any files in the format **YourStudentID_name.*** (e.g., 10123456_assignment2.pdf, 10123456_problem1.py and 10123456_README.md) and upload them to Canvas. Note that any code error would lead to 0 points for the respective problem. Required results should be shown clearly.

**Late submission**: 25 marks will be deducted for every 24 hours after the submission deadline.