Python Programming

A* Search, 8-Puzzle and CSP

Rathindra Nath Dutta

Senior Research Fellow
Advanced Computing & Microelectronics Unit
Indian Statistical Institute, Kolkata

October 03, 2023

A* Search

A simple implementation: a_star.py

Designing an 8-Puzzle Class

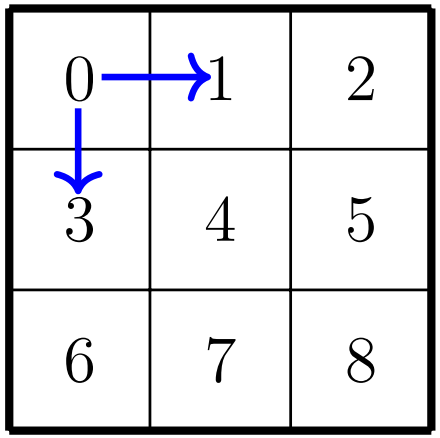
- State

```
class EightPuzzleState:
    def __init__(self, state):
        self.state = state
        self.blank_pos = state.index(0)
    ...
```

Designing an 8-Puzzle Class

- State
- Moves

```
class EightPuzzleState:
    ...
    possible_moves = {0: (1,3), 1: (0,2,4), ..., 8: (5,7)}
    ...
```



Designing an 8-Puzzle Class

- State
- Moves

```
class EightPuzzleState:
    ...
    def generate_next_states(self):
        valid_moves = possible_moves[self.blank_pos]
        next_states = []
        for pos in valid_moves:
            # swap blank with pos
            new_state = self.state.copy() # must be copied
            new_state[self.blank_pos], new_state[pos] =
            ↪  new_state[pos], new_state[self.blank_pos]
            next_states.append(EightPuzzleState(new_state))
        return next_states
    ...
```

Designing an 8-Puzzle Class

- State
- Moves
- Misc.

```
class EightPuzzleState:
    ...
    def __eq__(self, other):
        return self.state == other.state

    def __hash__(self):
        # return hash(tuple(self.state))
        return hash(str(self.state))
    ...
```

Designing an 8-Puzzle Class

- State
- Moves
- Misc.

```
class EightPuzzleState:
    ...
    def __repr__(self):
        # write your fancy code here
        # May use the box drawing characters

    def __str__(self):
        return self.__repr__() # reusing
```

See: https://en.wikipedia.org/wiki/Box-drawing_character

8-Puzzle: Manhattan Distance Heuristic

```
def Manhattan_dist(x1, y1, x2, y2):
    return abs(x1 - x2) + abs(y1 - y2)
```

8-Puzzle: Manhattan Distance Heuristic

```
def Manhattan_dist(x1, y1, x2, y2):
    return abs(x1 - x2) + abs(y1 - y2)
```

```
def eight_puzzle_Manhattan_dist(state1, state2):
    sum = 0
    for i in range(1, 9):
        position1 = state1.state.index(i)
        position2 = state2.state.index(i)
        sum += Manhattan_dist(position1 // 3, position1 %3,
        ↪  position2 // 3, position2 % 3)
    return sum
```

8-Puzzle: Solvability

- Fix any goal state
- Not all start state can be solved

8-Puzzle: Solvability

- Fix any goal state
- Not all start state can be solved
- We can easily determine solvability without actually solving it!

8-Puzzle: Solvability

- Fix any goal state
- Not all start state can be solved
- We can easily determine solvability without actually solving it!

Theorem (Solvability of 8-Puzzle)

A given start state s is solvable against a goal state g if and only if the number of inversions (parity) between s and g is even

8-Puzzle: Solvability

- Fix any goal state
- Not all start state can be solved
- We can easily determine solvability without actually solving it!

Theorem (Solvability of 8-Puzzle)

A given start state s is solvable against a goal state g if and only if the number of inversions (parity) between s and g is even

Read more here:

https://puzzling.stackexchange.com/a/52111

https://math.stackexchange.com/questions/293527

https://en.wikipedia.org/wiki/15_Puzzle#Solvability

Solving 8-Puzzle using A*

A simple implementation: eight_puzzle.py

Constraint Satisfaction Problem

- There are many CSP solvers for Python
- We are going to explore the `python-constraint`[1] library
- It can be installed with

```
pip install python-constraint
```

[1] Documentation: https://python-constraint.github.io/python-constraint/

A Simple CSP Formulation

Find all Pythagorean triplets between 1 and 20

A Simple CSP Formulation

Find all Pythagorean triplets between 1 and 20

$$\begin{array}{ll} & \text{Find all } a,\ b \text{ and } c \\ \text{such that:} & a^2 + b^2 = c^2 \text{ and } a, b, c \in \{1, 2, \ldots, 20\} \end{array}$$

A Simple CSP Formulation

Find all Pythagorean triplets between 1 and 20

$$\text{Find all } a,\ b \text{ and } c$$

$$\text{such that:} \quad a^2 + b^2 = c^2 \text{ and } a, b, c \in \{1, 2, \ldots, 20\}$$

$$\text{also we may want} \quad a \leq b \leq c \quad \text{to avoid repetitions}$$

A Simple CSP Formulation

```
import constraint as csp

problem = csp.Problem() # new csp problem

problem.addVariable(variable='a', domain=range(1,21))
problem.addVariable(variable='b', domain=range(1,21))
problem.addVariable(variable='c', domain=range(1,21))

c1 = lambda a, b, c: a*a + b*b == c*c
problem.addConstraint(constraint=c1, variables=['a', 'b', 'c'])

# ordered triplets: a <= b <= c
c2 = lambda a, b: a <= b
problem.addConstraint(constraint=c2, variables=['a', 'b'])

print(problem.getSolution()) # get one of the solutions as dict
print(problem.getSolutions()) # list of all possible solutions
```

Another CSP Example

Find a Vertex Cover V_C of a Graph $G = (V, E)$

Another CSP Example

Find a Vertex Cover V_C of a Graph $G = (V, E)$

Clearly we have: $V_C \subseteq V$

Another CSP Example

Find a Vertex Cover V_C of a Graph $G = (V, E)$

Clearly we have: $V_C \subseteq V$

Let us define for each $v \in V$: $X_v = \begin{cases} 1 & \text{if we pick vertex } v \text{ into } V_C \\ 0 & \text{otherwise} \end{cases}$

Another CSP Example

Find a Vertex Cover V_C of a Graph $G = (V, E)$

Clearly we have: $V_C \subseteq V$

Let us define for each $v \in V$: $X_v = \begin{cases} 1 & \text{if we pick vertex } v \text{ into } V_C \\ 0 & \text{otherwise} \end{cases}$

Constraint: for each edge $(u, v) \in E$ we must have $X_u + X_v \geq 1$

Another CSP Example

Find a Vertex Cover V_C of a Graph $G = (V, E)$

Clearly we have: $V_C \subseteq V$

Let us define for each $v \in V$: $X_v = \begin{cases} 1 & \text{if we pick vertex } v \text{ into } V_C \\ 0 & \text{otherwise} \end{cases}$

Constraint: for each edge $(u, v) \in E$ we must have $X_u + X_v \geq 1$

Finding one V_c is trivial: just take $V_C = V$

Another CSP Example

Find a Vertex Cover V_C of a Graph $G = (V, E)$

Clearly we have: $V_C \subseteq V$

Let us define for each $v \in V$: $X_v = \begin{cases} 1 & \text{if we pick vertex } v \text{ into } V_C \\ 0 & \text{otherwise} \end{cases}$

Constraint: for each edge $(u, v) \in E$ we must have $X_u + X_v \geq 1$

Finding one V_c is trivial: just take $V_C = V$

Challenge: find a V_C having minimum cardinality (NP-Complete)

Another CSP Example: Vertex Cover

```
import constraint as csp
import networkx as nx
def get_min_vertex_cover_size(G):
    problem = csp.Problem()
    for v in G.nodes:
        problem.addVariable(variable=v, domain=[0, 1])
    for u,v in G.edges:
        problem.addConstraint(constraint=lambda u,v: u + v >= 1,
        ↪  variables=[u, v])

    min_size = G.number_of_nodes()
    for sol in problem.getSolutions():
        size = sum( sol.values() )
        if size < min_size: min_size = size
    return min_size

print( get_min_vertex_cover_size( nx.complete_graph(5) ) )
print( get_min_vertex_cover_size( nx.star_graph(6) ) )
```

[1] Explore: https://networkx.org/documentation/stable/tutorial.html