Hyun-Joon Yang

yanghyun@usc.edu

CSCI 360

Lab 2 Extra Credit

In [1]:

```
from heapq import heappush, heappop

def get_state(stack):
    state = ''
    for i in stack.order:
        state += str(i)
    for i in stack.orientations:
        state += str(i)
    return state

class Node:
    count = 0

    def __init__(self, g, stack, parent, flip):
        self.g = g
        self.name = Node.count
        self.stack = stack
        self.state = get_state(stack)
        self.parent = parent
        self.flip = flip
        Node.count += 1

    def get_f(self):
        return self.g + self.get_h()

    def get_wf(self, epsilon, N):
        return self.g + (self.get_w(epsilon, N) * self.get_h())

    def get_h(self):
        def is1(order):
            diff = order[0] - order[1]
            return not (diff == 1 or diff == -1)
        def is2(orient):
            return orient[0] != orient[1]
        def is3(pair, orient):
            return (pair[0] + 1 != pair[1]) and (orient[0] and orient[1])
        def is4(pair, orient):
            return (pair[0] + 1 == pair[1]) and (not orient[0] and not orient[1])

        h = 0
        for i in range(self.stack.num_books-1):
            order = self.stack.order[i:i+2]
            orient = self.stack.orientations[i:i+2]
            if is1(order) or is2(orient) or is3(order, orient) or is4(order, orient):
                h += 1
        return h

    def get_w(self, epsilon, N):
        return 1 + epsilon - ((epsilon * self.g) / N)
```

In [2]:

```
def a_star_search(stack):

    flip_sequence = []

    # --- v ADD YOUR CODE HERE v --- #
    pq = []
    visited = set({})
    visit_order = []

    node = Node(0, stack, None, 0)
    visited.add(node.state)
    entry = [node.get_f(), node.name, node]
    heappush(pq, entry)

    while len(pq) > 0:
        # get next node from fringe
        entry = heappop(pq)
        node = entry[-1]
        visit_order.append(node)

        # check if goal node & trace
        if node.stack.check_ordered():
            while node.parent is not None:
                flip_sequence.append(node.flip)
                node = node.parent
            Node.count = 0
            return flip_sequence[::-1], visit_order

        # enumerate child nodes
        for i in range(stack.num_books):
            cpy = node.stack.copy()
            cpy.flip_stack(i+1)
            # add to fringe
            state = get_state(cpy)
            new_node = Node(node.g+1, cpy, node, i+1)
            if state not in visited:
                visited.add(new_node.state)
                entry = [new_node.get_f(), new_node.name, new_node]
                heappush(pq, entry)

    return flip_sequence, visit_order
    # ---------------------------- #
```

In [3]:

```
def weighted_a_star_search(stack, epsilon=None, N=1):
    # Weighted A* is extra credit

    flip_sequence = []

    # --- v ADD YOUR CODE HERE v --- #
    pq = []
    visited = set({})
    visit_order = []

    node = Node(0, stack, None, 0)
    visited.add(node.state)
    entry = [node.get_wf(epsilon, N), node.name, node]
    heappush(pq, entry)

    while len(pq) > 0:
        # get next node from fringe
        entry = heappop(pq)
        node = entry[-1]
        visit_order.append(node)

        # check if goal node & trace
        if node.stack.check_ordered():
            while node.parent is not None:
                flip_sequence.append(node.flip)
                node = node.parent
            Node.count = 0
            return flip_sequence[::-1], visit_order

        # enumerate child nodes
        for i in range(stack.num_books):
            cpy = node.stack.copy()
            cpy.flip_stack(i+1)
            # add to fringe
            state = get_state(cpy)
            new_node = Node(node.g+1, cpy, node, i+1)
            if state not in visited:
                visited.add(new_node.state)
                entry = [node.get_wf(epsilon, N), new_node.name, new_node]
                heappush(pq, entry)

    return flip_sequence, visit_order
    # ---------------------------- #
```

In [4]:

```
from lab2_utils import TextbookStack

def orderHelper(n, numbers, order, orders):
    if len(order) == n:
        orders.append(order)
        return
    for number in numbers:
        cpy = [i for i in order]
        cpy.append(number)
        orderHelper(n, set({i for i in numbers if i != number}), cpy, orders)

def generateOrder(n):
    orders = []
    numbers = set({})
    for i in range(n):
        numbers.add(i)
    orderHelper(n, numbers, [], orders)
    return orders

def orientationHelper(n, numbers, orientation, orientations):
    if len(orientation) == n:
        orientations.append(orientation)
        return
    for number in numbers:
        cpy = [i for i in orientation]
        cpy.append(number)
        orientationHelper(n, numbers, cpy, orientations)

def generateOrientation(n):
    orientations = []
    numbers = set({0, 1})
    orientationHelper(n, numbers, [], orientations)
    return orientations

def generateStacks(n):
    textbooks = []
    orders = generateOrder(n)
    orientations = generateOrientation(n)
    for i in orders:
        for j in orientations:
            textbooks.append(TextbookStack(i, j))
    return textbooks
```

In [5]:

```
from math import factorial
import numpy as np

table_flips = np.zeros((8,3))
table_nodes = np.zeros((8,3))

for m in range(1, 9):
    textbooks = generateStacks(m)
    print('m:', m)
    print('expected:', 2**m * factorial(m))
    print('output:', len(textbooks))
    print('-'*20)

    flips_a = 0
    flips_wa = 0
    nodes_a = 0
    nodes_wa = 0

    for textbook in textbooks:
        seq_a, order_a = a_star_search(textbook)
        seq_wa, order_wa = weighted_a_star_search(textbook, epsilon=1, N=2*m)
        flips_a += len(seq_a)
        flips_wa += len(seq_wa)
        nodes_a += len(order_a)
        nodes_wa += len(order_wa)

    table_flips[m-1, 0] = m
    table_nodes[m-1, 0] = m

    table_flips[m-1, 1] = flips_a / len(textbooks)
    table_nodes[m-1, 1] = nodes_a / len(textbooks)

    table_flips[m-1, 2] = flips_wa / len(textbooks)
    table_nodes[m-1, 2] = nodes_wa / len(textbooks)
```

```
m: 1
expected: 2
output: 2
--------------------
m: 2
expected: 8
output: 8
--------------------
m: 3
expected: 48
output: 48
--------------------
m: 4
expected: 384
output: 384
--------------------
m: 5
expected: 3840
output: 3840
--------------------
m: 6
expected: 46080
output: 46080
--------------------

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-5-b70b8b611515> in <module>
     18 
     19     for textbook in textbooks:
---> 20         seq_a, order_a = a_star_search(textbook)
     21         seq_wa, order_wa = weighted_a_star_search(textbook, epsilon=1, N=2*m)
     22         flips_a += len(seq_a)

<ipython-input-2-ed484becf2f7> in a_star_search(stack)
```

```
     29         # enumerate child nodes
     30         for i in range(stack.num_books):
---> 31             cpy = node.stack.copy()
     32             cpy.flip_stack(i+1)
     33             # add to fringe

~\OneDrive\바탕 화면\Classes\FALL 2020\CSCI 360\Labs\Lab2\l2\lab2_utils.py in copy(self)
     31
     32     def copy(self):
---> 33         return TextbookStack(self.order, self.orientations)
     34
     35     def __eq__(self, other):

~\OneDrive\바탕 화면\Classes\FALL 2020\CSCI 360\Labs\Lab2\l2\lab2_utils.py in __init__(self, initial_order, initial_orientations)
     14             assert a == 1 or a == 0
     15
---> 16         self.order = np.array(initial_order)
     17         self.orientations = np.array(initial_orientations)
     18

KeyboardInterrupt:
```

In [6]:

```python
print("Number of Flips")
print("  m          A*         Weighted A*")
print('-'*38)
print(table_flips)
print('\n')
print("Number of Nodes Visited")
print("  m          A*         Weighted A*")
print('-'*38)
print(table_nodes)
```

```
Number of Flips
  m          A*         Weighted A*
--------------------------------------
[[1.         0.5        0.5       ]
 [2.         2.         2.        ]
 [3.         3.4375     3.52083333]
 [4.         4.80989583 5.07552083]
 [5.         6.15       6.63203125]
 [0.         0.         0.        ]
 [0.         0.         0.        ]
 [0.         0.         0.        ]]


Number of Nodes Visited
  m          A*         Weighted A*
--------------------------------------
[[  1.           1.5          1.5       ]
 [  2.           4.125        4.125     ]
 [  3.          13.08333333  11.1875    ]
 [  4.          46.69791667  26.2109375 ]
 [  5.         177.9296875   53.11484375]
 [  0.           0.           0.        ]
 [  0.           0.           0.        ]
 [  0.           0.           0.        ]]
```

It seems while weighted A* seems to visit significantly less number of nodes than A* as m increases, its solutions require more number of flips, consistent with its compromise in optimality.