import numpy as np

import matplotlib.pyplot as plt

import matplotlib.patches as patches

from matplotlib.patches import Patch, FancyArrowPatch

from matplotlib.lines import Line2D

from itertools import combinations

def rule\_based\_actions(A,B,C,D):

acts = []

p1 = (A, B)

p2 = (C, D)

if p1 == (3,3) and p2 == (3,3):

acts.append((None, ('A','C')))

acts.append((None, ('B','C')))

if p1 == (3,3) and p2 == (3,2):

acts.append(('swap\_AC', ()))

acts.append(('swap\_BD', ()))

if p1 == (3,3) and p2 in {(3,1),(3,0)}:

acts.append(('swap\_BC', ('B',)))

if p1 == (3,3) and p2 == (2,3):

acts.append(('swap\_BD', ('A','D')))

if p1 == (3,3) and p2 in {(1,3),(0,3)}:

acts.append(('swap\_BD', ('B',)))

if p1 == (3,2) and p2 == (3,3):

acts.append(('swap\_BC', ('A','C')))

if p1 == (2,3) and p2 in {(3,2),(3,1),(3,0)}:

acts.append((None, ('A','C')))

if p1 == (2,3) and p2 in {(2,3),(1,3),(0,3)}:

acts.append((None, ('A','D')))

if p1 == (3,1) and p2 == (3,3):

acts.append(('swap\_BC', ('D',)))

if p1 == (1,3) and p2 in {(3,2),(3,1)}:

acts.append((None, ('A','C')))

if p1 == (3,1) and p2 in {(3,0),(2,2),(2,1),(2,0)}:

acts.append(('swap\_BC', ()))

if p1 == (3,1) and p2 in {(2,3),(1,3)}:

acts.append((None, ('A','D')))

if p1 == (3,1) and p2 in {(1,2),(0,3),(0,2)}:

acts.append(('swap\_BD', ()))

if p1 == (3,0) and p2 == (3,3):

acts.append(('swap\_BC', ('D',)))

if p1 == (3,0) and p2 == (3,2):

acts.append((None, ('A','C')))

if p1 == (3,0) and p2 in {(3,1),(3,0),(2,2),(2,1),(2,0),(1,1),(1,0)}:

acts.append(('swap\_BC', ()))

if p1 == (3,0) and p2 in {(1,3),(1,2),(0,3),(0,2),(0,1)}:

acts.append(('swap\_BD', ()))

if p1 == (3,0) and p2 == (2,3):

acts.append((None, ('A','D')))

if p1 == (2,3) and p2 == (3,3):

acts.append(('swap\_BD', ('B','C')))

if p1 == (2,3) and p2 in {(3,2),(3,1),(3,0)}:

acts.append(('swap\_BC', ()))

if p1 == (2,3) and p2 in {(2,3),(1,3),(0,3)}:

acts.append((None, ('B','D')))

if p1 in {(2,2),(2,1),(2,0)} and p2 in {(3,1),(3,0)}:

acts.append(('swap\_BC', ()))

if p1 in {(2,2),(2,1),(2,0)} and p2 in {(1,3),(0,3)}:

acts.append(('swap\_BD', ()))

if p1 == (1,3) and p2 == (3,3):

acts.append(('swap\_BD', ('D',)))

if p1 == (1,3) and p2 in {(3,2),(3,1)}:

acts.append((None, ('B','C')))

if p1 == (1,3) and p2 in {(3,0),(2,2),(2,1),(2,0)}:

acts.append(('swap\_BD', ()))

if p1 == (1,3) and p2 in {(2,3),(1,3)}:

acts.append((None, ('B','D')))

if p1 == (1,3) and p2 in {(1,2),(0,3),(0,2)}:

acts.append(('swap\_BC', ()))

if p1 == (1,2) and p2 in {(3,1),(3,0)}:

acts.append(('swap\_BD', ()))

if p1 == (1,2) and p2 in {(1,3),(0,3)}:

acts.append(('swap\_BC', ()))

if p1 in {(1,1),(1,0)} and p2 == (3,0):

acts.append(('swap\_BC', ()))

if p1 in {(1,1),(1,0)} and p2 == (0,3):

acts.append(('swap\_BD', ()))

if p1 == (0,3) and p2 == (3,3):

acts.append(('swap\_BD', ('D',)))

if p1 == (0,3) and p2 == (3,2):

acts.append((None, ('B','C')))

if p1 == (0,3) and p2 in {(3,1),(3,0),(2,2),(2,1),(2,0),(1,1),(1,0)}:

acts.append(('swap\_BD', ()))

if p1 == (0,3) and p2 in {(1,3),(1,2),(0,3),(0,1),(0,2)}:

acts.append(('swap\_BC', ()))

if p1 == (0,2) and p2 in {(3,1),(3,0)}:

acts.append(('swap\_BD', ()))

if p1 == (0,2) and p2 in {(1,3),(0,3)}:

acts.append(('swap\_BC', ()))

if p1 == (0,1) and p2 == (3,0):

acts.append(('swap\_BD', ()))

if p1 == (0,1) and p2 == (0,3):

acts.append(('swap\_BC', ()))

unique = []

for a in acts:

if a not in unique:

unique.append(a)

return unique

# رسم سیاست

color\_map = {

(): 'white',

('A',): '#e6e6e6', ('B',): '#f4cccc', ('C',): '#cfe2f3', ('D',): '#d9ead3',

('A','B'): '#b6d7a8', ('A','C'): '#f9cb9c', ('A','D'): '#fff2cc',

('B','C'): '#9fc5e8', ('B','D'): '#ffe599', ('C','D'): '#6fa8dc',

('A','B','C'): '#e69138', ('A','B','D'): '#93c47d', ('A','C','D'): '#b4a7d6',

('B','C','D'): '#ffd966', ('A','B','C','D'): '#999999'

}

fig, ax = plt.subplots(figsize=(16, 12))

for i in range(16):

for j in range(16):

s = i \* 16 + j

acts = feasible\_actions[s]

if acts:

realloc, rep = acts[policy[s]]

else:

realloc, rep = (None, ())

rep = tuple(sorted(rep))

rect = patches.Rectangle((j, 15 - i), 1, 1,

facecolor=color\_map.get(rep, 'white'),

edgecolor='black', linewidth=0.6)

ax.add\_patch(rect)

x, y = j + 0.5, 15 - i + 0.5

if realloc == 'swap\_BC':

arrow = FancyArrowPatch((x, y + 0.22), (x, y - 0.22),

arrowstyle='<->', mutation\_scale=8,

color='red', lw=1.2, zorder=3)

ax.add\_patch(arrow)

ax.text(x, y + 0.34, 'B', ha='center', va='center', fontsize=7, color='red')

ax.text(x, y - 0.34, 'C', ha='center', va='center', fontsize=7, color='red')

elif realloc == 'swap\_BD':

arrow = FancyArrowPatch((x - 0.22, y), (x + 0.22, y),

arrowstyle='<->', mutation\_scale=8,

color='red', lw=1.2, zorder=3)

ax.add\_patch(arrow)

ax.text(x - 0.34, y, 'B', ha='center', va='center', fontsize=7, color='red')

ax.text(x + 0.34, y, 'D', ha='center', va='center', fontsize=7, color='red')

elif realloc == 'swap\_AC':

# نشان‌دادن A در بالا-چپ و C در پایین-راستِ سلول

ax.text(x - 0.26, y + 0.26, 'A', ha='center', va='center', fontsize=7, color='red')

ax.text(x + 0.26, y - 0.26, 'C', ha='center', va='center', fontsize=7, color='red')

# یک خط مورب کوچک برای نمایش بازتخصیص

arrow = FancyArrowPatch((x - 0.18, y + 0.18), (x + 0.18, y - 0.18),

arrowstyle='<->', mutation\_scale=6, color='red', lw=1.0, zorder=3)

ax.add\_patch(arrow)

elif realloc == 'swap\_AC\_BD':

# دو جفت را با متن ساده نشان می‌دهیم

ax.text(x - 0.26, y + 0.26, 'A', ha='center', va='center', fontsize=7, color='red')

ax.text(x + 0.26, y - 0.26, 'C', ha='center', va='center', fontsize=7, color='red')

ax.text(x - 0.26, y - 0.26, 'B', ha='center', va='center', fontsize=7, color='red')

ax.text(x + 0.26, y + 0.26, 'D', ha='center', va='center', fontsize=7, color='red')

arrow1 = FancyArrowPatch((x - 0.18, y + 0.18), (x + 0.18, y - 0.18),

arrowstyle='<->', mutation\_scale=6, color='red', lw=1.0, zorder=3)

arrow2 = FancyArrowPatch((x - 0.18, y - 0.18), (x + 0.18, y + 0.18),

arrowstyle='<->', mutation\_scale=6, color='red', lw=1.0, zorder=3)

ax.add\_patch(arrow1)

ax.add\_patch(arrow2)

# راهنما

legend\_labels = [

('no replace', ()),

('replace A', ('A',)),

('replace B', ('B',)),

('replace C', ('C',)),

('replace D', ('D',)),

('replace A and B', ('A','B')),

('replace A and C', ('A','C')),

('replace A and D', ('A','D')),

('replace B and C', ('B','C')),

('replace B and D', ('B','D')),

('replace C and D', ('C','D')),

]

legend\_elements = [Patch(facecolor=color\_map[k], label=lbl, edgecolor='black') for lbl, k in legend\_labels]

legend\_elements += [

Line2D([], [], color='red', lw=1.5, marker=None, label='B ↔ C (realloc)'),

Line2D([], [], color='red', lw=1.5, marker=None, label='B ↔ D (realloc)'),

Line2D([], [], color='red', lw=1.5, marker=None, label='A ↔ C (realloc)')

]

ax.legend(handles=legend\_elements, loc='center left', bbox\_to\_anchor=(1.02, 0.5),

fontsize=10, title='Actions')

labels = [f"({i//4},{i%4})" for i in range(16)]

ax.set\_xlim(0, 16)

ax.set\_ylim(0, 16)

ax.set\_aspect('equal')

ax.set\_xticks(np.arange(0.5, 16.5, 1))

ax.set\_xticklabels(labels, rotation=45, ha='right', fontsize=8)

ax.set\_yticks(np.arange(0.5, 16.5, 1))

ax.set\_yticklabels(labels[::-1], fontsize=8)

ax.set\_xlabel('Pair 2=(C, D)', fontsize=14, fontweight='bold')

ax.set\_ylabel('Pair 1=(A, B)', fontsize=14, fontweight='bold')

ax.set\_title('Policy Iteration — Rule-based Realloc', fontsize=16, fontweight='bold')

plt.tight\_layout()

plt.savefig("policy\_iteration.png", dpi=300, bbox\_inches='tight')

plt.show()