

C Experimental Details

1 Import Modules

```
[1]: import numpy as np
from fourrooms import Fourrooms
from IPython.display import clear_output
from aoac_tabular import *
import matplotlib.pyplot as plt
from visualize import Visualization
```

2 HyperParameters

```
[2]: # Replace the command line argparse
class Arguments:
    def __init__(self):
        # Numbers
        self.nepisodes=3000
        self.nsteps=2000
        self.noptions=2

        # Learning Rates
        self.lr_term=0.25
        self.lr_intra=0.25
        self.lr_critic=0.5
        self.lr_criticA=0.5
        self.lr_attend=0.01

        # Environment Parameters
        self.discount=0.99
        self.punishEachStep = True
        self.modified = True

        # Attention Parameters
        self.h_learn = True
        self.normalize = True

        # Policy Parameters
        self.epsilon=1e-1
        self.temp=2.

        # Objective Parameters
        self.wo1 = 0.1    #q
        self.wo2 = 1.     #cosim
        self.wo3 = 0.     #entropy
        self.wo4 = 0.     #size

        # Randomness Parameters
        self.seed=2222
        self.seed_startstate=1111
```

```

        # Display Parameters
        self.showMap = False
        self.showAttention = False
        self.showOptPref = False
        self.showFrequency = 10

        # Other Parameters
        self.baseline=True
        self.dc = 2.

args = Arguments()

```

3 Run

3.1 Set up

```

[3]: rng = np.random.RandomState(args.seed)
env = Fourrooms(args.seed_startstate, args.punishEachStep, args.modified)

features = Tabular(env.observation_space)
nfeatures, nactions = len(features), env.action_space

viz = Visualization(env, args, nactions)

```

3.2 Main loop

```

[ ]: # Set up classes
policy_over_options = P00(rng, nfeatures, args)
CoSimObj.reset()
options = [Option(rng, nfeatures, nactions, args, policy_over_options, i) for i in
           range(args.noptions)]

# Loop through games
for episode in range(args.nepisodes):
    # Initial state
    return_per_episode = 0.0
    observation = env.reset()
    phi = features(observation)
    option = policy_over_options.sample(phi)
    action = options[option].sample(phi)
    traject = [[phi,option],[phi,option],action]
    viz.resetMap(phi)

    # Reset record
    cumreward = 0.
    duration = 1
    option_switches = 0
    avgduration = 0.

    # Loop through frames in 1 game

```

```

for step in range(args.nsteps):
    # Collect feedback from environment
    observation, reward, done, _ = env.step(action)
    phi = features(observation)
    return_per_episode += pow(args.discount, step) * reward

    # Render
    if args.showMap and episode % 100 == 99:
        clear_output(wait=True)
        viz.showMap(phi, option)

    # Store option index
    last_option = option

    # Check termination
    termination = options[option].terminate(phi, value=True)
    if options[option].terminate(phi):
        option = policy_over_options.sample(phi)
        option_switches += 1
        avgduration += (1./option_switches)*(duration - avgduration)
        duration = 1

    # Record into trajectory
    traject[0] = traject[1]
    traject[1] = [phi, option]
    traject[2] = action

    # Sample next action
    action = options[option].sample(phi)

    # Policy Evaluation + Policy Improvement
    baseline = policy_over_options.value(traject[0][0], traject[0][1])
    advantage = policy_over_options.advantage(phi, last_option)
    options[last_option].update(traject, reward, done, phi, last_option,
    termination, baseline, advantage)
    policy_over_options.update(traject, reward, done, termination)

    # End of frame
    cumreward += reward
    duration += 1
    if done:
        break

    # Attention graph
    if episode % args.showFrequency == 0:
        if args.showAttention:
            clear_output(wait=True)
            viz.showAttention(options)
            print(options[0].policy.attention.weights)
        if args.showOptPref:
            clear_output(wait=True)
        print('Episode {} steps {} cumreward {} avg. duration {} switches {}'.
        format(episode, step, cumreward, avgduration, option_switches))

```

4 Visualization

4.1 Simulate an episode

```
[5]: states = np.zeros((13,13), dtype="int")
occupancy = env.occupancy.astype('float64')
s=0
for i in range(13):
    for j in range(13):
        if occupancy[i,j] == 0:
            states[i,j] = s
            s+=1
print(states)
```

```
[ [ 0  0  0  0  0  0  0  0  0  0  0  0  0]
  [ 0  0  1  2  3  4  0  5  6  7  8  9  0]
  [ 0 10 11 12 13 14  0 15 16 17 18 19  0]
  [ 0 20 21 22 23 24 25 26 27 28 29 30  0]
  [ 0 31 32 33 34 35  0 36 37 38 39 40  0]
  [ 0 41 42 43 44 45  0 46 47 48 49 50  0]
  [ 0  0 51  0  0  0  0 52 53 54 55 56  0]
  [ 0 57 58 59 60 61  0  0  0 62  0  0  0]
  [ 0 63 64 65 66 67  0 68 69 70 71 72  0]
  [ 0 73 74 75 76 77  0 78 79 80 81 82  0]
  [ 0 83 84 85 86 87 88 89 90 91 92 93  0]
  [ 0 94 95 96 97 98  0 99 100 101 102 103  0]
  [ 0  0  0  0  0  0  0  0  0  0  0  0  0]]
```

```
[6]: startState = 57
# Simulation
observation = env.reset(startState)
viz.resetMap(phi)

option = policy_over_options.sample(phi)
action = options[option].sample(phi)

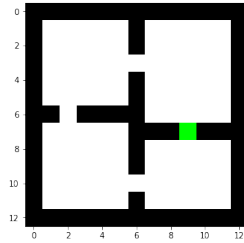
for step in range(args.nsteps):
    observation, reward, done, _ = env.step(action)
    phi = features(observation)

    #render
    clear_output(wait=True)
    viz.showMap(phi, option)

    if options[option].terminate(phi):
        option = policy_over_options.sample(phi)

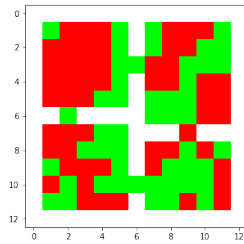
    action = options[option].sample(phi)

    if done:
        break
```

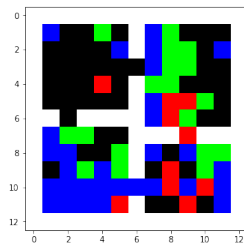


4.2 Display action and option preference in each state

```
[7]: # Display option preference
viz.showPref(policy_over_options.weights)
```



```
[8]: opt = 0
# Display action preference for opt
viz.showPref(options[opt].weights)
```



4.3 Display Attention

```
[9]: viz.showAttention(options)
```

