COGS 182 Project 2 | Schedule Optimization

Checkpoint 2

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import sys
from mpl_toolkits import mplot3d
import seaborn as sns
from tqdm.auto import tqdm
import pickle
```

List of Possible Topics & Actions (Hour Allocations to Each Topic) & Resultant States

In [2]:
```
possible_topics=['topology',
                 'Japanese',
                 'webtooning',
                 'physics',
                 'video-editing',
                 'graphic design',
                 'animation',
                 'psych/linguistics',
                 'R/statistical packages',
                 'front-end coding']

#0-12 = allot hrs, 7 = commit to schedule
possible_actions = [0,1,3,5,7]
```

In [3]:

```
r_times = []
for remaining_time in np.arange(25):
    for action in [0,1,3,5]:
        for org_hrs in [0,1,2,3,4]:
            for lab_hrs in [0, 1,2]:
                r_time = remaining_time - action - org_hrs - lab_hrs
                r_times.append(r_time)

possible_remaining_times = sorted(np.unique(r_times))


terminal_states = []
for r_time in possible_remaining_times:
    terminal_state = [r_time, 0]
    terminal_states.append(terminal_state)

possible_states = [0]
for r_time in possible_remaining_times:
        for len_schedule in np.arange(len(possible_topics) +1):
            for topic in possible_topics:
                state = [r_time, len_schedule, topic]
                if state[0] > 0:
                    possible_states.append(state)

# len(terminal_states),len(possible_states)

pos_states = [0]
for state_indx, state in enumerate(possible_states[1:]):
    r_time, len_schd, topic = state
    no_topic_state = [r_time, len_schd]
    if no_topic_state not in pos_states:
        pos_states.append(no_topic_state)
```

Factors (i.e. schedule fulfillment, coherence) to take into account when calculating the rewards

In [4]:

```
#mutually reinforcing subjects (not really accurate to real life, but it suffices)
related_topics = {'topology': ['physics', 'R/statistical packages', 'graphic design'],
                  'Japanese': ['linguistics', 'animation', 'webtooning'],
                  'webtooning': ['animation', 'Japanese', 'graphic design'],
                  'physics': ['topology','chemistry', 'R/statistical packages'],
                  'video-editing': ['animation', 'content creation', 'webtooning'],
                  'graphic design': ['webtooning', 'front-end coding'],
                  'animation': ['webtooning', 'video-editing', 'graphic design', 'content creation'],
                  'psych/linguistics': ['Japanese', 'R/statistical packages'],
                  'R/statistical packages': ['educational psychology', 'physics'],
                  'front-end coding': ['graphic design', 'animation', 'content creation']}

# personal ratings of [frustration/learning curve, intrigue, applicability] by topic
fulfillment_factors = {'topology': [1, 1, 0.6],
                       'Japanese': [0.7, 1, 1],
                        'webtooning': [0.5, 0.5, 0.4],
                        'physics': [1, 0.9, .6],
                        'video-editing': [0.8, 0.4, 1],
                        'graphic design': [0.5, 0.3, 0.6],
                        'animation': [1, 0.9, 0.7],
                        'psych/linguistics': [0.6, 0.7, 0.7],
                        'R/statistical packages': [0.5, 0.4, 1],
                        'front-end coding': [0.6, 0.7, 1]}
fulfillment_scores = {}
for topic in fulfillment_factors:
    frustration, intrigue, applicability = fulfillment_factors[topic]

    #fulfillment factors scale positive rewards
    fulfillment_score = (0.25*frustration + 0.5*intrigue + 0.25*applicability)
    fulfillment_scores[topic] = fulfillment_score
```

Step function, take in remaining time (in a day), the schedule, the topic of consideration, and the action

- NOTE: for feasibility sake, the observed state is actually only the LENGTH of the current schedule, but the entire schedule is passed for the environment to do calculations

In [5]:
```
# state keeps track of remaining free time in a day, the length of the schedul
e, and the current topic
    # length of schedule instead of actual schedule, since that's too many dif
ferent states...

def step(remaining_time, schedule, topic, action):

    state = [remaining_time, len(schedule)]
    state_indx = pos_states.index(state)

    if action != 7:
        if action != 0:
            remaining_time -= action
            schedule[topic] = action #then allot hours according to a policy?

        topic = np.random.choice(possible_topics)
        new_state = [remaining_time, len(schedule)]
        reward = 0


        if remaining_time < 6:
            # print("\033[1;31m  KAROSHI  \033[0m YOU DIED OF OVERWORK")
            # sys.stdout.flush()
            reward = -1
            new_state = 0

        return new_state, reward, schedule, topic


    # ENVIRONMENT GOES when commit to schedule
    elif action == 7:
        new_state, reward, schedule, remaining_time = env(remaining_time, sche
dule)

        return new_state, reward, schedule, remaining_time
```

When the environment goes,

it takes the remaining time and schedule, adds on random hours from existing commitments, calculates if the remaining time in a day allows for sleep, calculates the schedule fulfillment and modulates the fulfillment with bonuses for schedule coherence and having enough to sleep. The terminal state is returned [remaining_time, 0].

In [6]:
```
def env(remaining_time, schedule):

        existing_commitments = {'Orgs': np.random.choice(np.arange(1,5)), 'Lab
Stuff': np.random.choice(np.arange(1,3))}
    # factor in existing commitments
        remaining_time -= (existing_commitments['Orgs'] + existing_commitments
['LabStuff'])

        # KAROSHI AGAIN if forget about current commitments
        if remaining_time < 6:
            reward = -1

        #penalty for undercommitment (listlessness)
        elif remaining_time > 18:
            reward = -0.5

        else:
            total_fulfillment = 0

            # get average of fulfillments of all commitments in schedule
            for topic in schedule:
                if topic in ["Orgs", "LabStuff"]:
                    continue
                else:
                    total_fulfillment += fulfillment_scores[topic]
            total_fulfillment = np.mean(total_fulfillment)

            # factor in related subjects into fulfillment score (mutually rein
forcing)
            # if too many subjects at once, coherence turns into distraction f
actor, negative rate
            coherence = 1
            if len(schedule) <= 3:
                for topic in schedule:
                    for possible_topic in possible_topics:
                        if topic == possible_topic:
                            for related_topic in related_topics[topic]:
                                if related_topic in schedule:
                                    coherence += 0.05
            elif len(schedule) > 4: #too many topics
                coherence = -0.5

            # sleep and free time bonus (a rate)
            if remaining_time >= 8 and remaining_time <= 10:
                bonus = 1.1           # 10% bonus fulfillment
            else:
                bonus = 1             # no bonus


            # reward to return as a function of bonus, topic-based fulfillment
and coherence of schedule
            reward = (total_fulfillment)* ((bonus + coherence)/ 2)

        schedule['Orgs'] = existing_commitments['Orgs']
        schedule['LabStuff'] = existing_commitments['LabStuff']
```

```
        new_state = 0

        return new_state, reward, schedule, remaining_time
```

In [ ]:

Check that environment dynamics $p(s', r|s, a)$ work

Test environment dynamics using select inputs

Input 1: step(remaining_time=1, schedule={'topology':2}, topic = "Japanese", action= 3)

- Certain karoshi, because action > remaining_time
- If action is not 7 ('commit'), the returned schedule should NOT contain "Orgs" or "LabStuff," since these are added when the environment goes.

In [7]:

```
# certain karoshi - alloting more hours to a topic than you have time in a day
print("Certain Karoshi (action > remaining_time or remaining_time + 6):")

new_state1, reward1, schedule1, topic1 = step(remaining_time=1,
                                              schedule={'topology':2},
                                              topic = "Japanese",
                                              action= 3)
print(new_state1, reward1, schedule1, topic1)
```

```
Certain Karoshi (action > remaining_time or remaining_time + 6):
0 -1 {'topology': 2, 'Japanese': 3} animation
```

Input 2: step(remaining_time=10, schedule={}, topic = "Japanese", action= 7)

- When the action is 7, the env goes, returning a schedule with "Orgs" and "LabStuff" and the terminal state, (remaining_time, 0])
- Uncertain Karoshi: Since the input schedule is empty but the hours allocated to "Orgs" and "LabStuff" range between 0 and 8 hours total, should see karoshi some of the time, i.e. if (Orgs + LabStuff - remaining_time) does not leave enough time for sleep (6 hours).

In [ ]:

In [8]:
```
print("\nUncertain Karoshi:")
# uncertain karoshi > if action is 7, then the environment should return the t
erminal state and remaining_time
#  should decrease, due to Orgs and Labstuff hours
is_karoshi = 0
for call in np.arange(10):
    new_state2, reward2, schedule2, remaining_time2 = step(remaining_time=10,
                                                          schedule={},
                                                          topic = "Japanese",
                                                          action= 7)
    call = [remaining_time2, reward2, schedule2]
    if reward2 == -1:
        is_karoshi += 1
    print(call)
print("Observed Karoshi Rate:", (is_karoshi / 10) *100, "%")
```

```
Uncertain Karoshi:
[6, 0.0, {'Orgs': 3, 'LabStuff': 1}]
[7, 0.0, {'Orgs': 1, 'LabStuff': 2}]
[5, -1, {'Orgs': 4, 'LabStuff': 1}]
[5, -1, {'Orgs': 3, 'LabStuff': 2}]
[8, 0.0, {'Orgs': 1, 'LabStuff': 1}]
[8, 0.0, {'Orgs': 1, 'LabStuff': 1}]
[5, -1, {'Orgs': 3, 'LabStuff': 2}]
[6, 0.0, {'Orgs': 2, 'LabStuff': 2}]
[5, -1, {'Orgs': 4, 'LabStuff': 1}]
[6, 0.0, {'Orgs': 3, 'LabStuff': 1}]
Observed Karoshi Rate: 40.0 %
```

In [9]:
```
is_karoshi = 0
for call in np.arange(10000):
    new_state2, reward2, schedule2, remaining_time2 = step(remaining_time=10,
                                                          schedule={},
                                                          topic = "Japanese",
                                                          action= 7)
    call = [remaining_time2, reward2, schedule2]
    if reward2 == -1:
        is_karoshi += 1
#     print(call)
print("Observed Karoshi Rate:", (is_karoshi / 10000) *100, "%")
```

```
Observed Karoshi Rate: 36.86 %
```

Input 3: step(remaining_time=24, schedule={'animation': 3, 'Japanese': 3}, topic = "Japanese", action= 7)

- & Input 4: step(remaining_time=24, schedule={'animation': 3, 'physics': 3}, topic = "Japanese", action= 7)
  - Rewards of coherent schedules (Input3) should be higher than noncoherent schedules (Input4)
  - UNLESS accounting for coheren schedules that UNDERCOMMIT

In [ ]:
```
```

```
In [10]: # rewards of coherent schedule should be higher than reward of noncoherent sch
edule
# unless it's org hours allow for more productivity
coherent_more = 0
for call in np.arange(10000):
    new_state3, reward3, schedule3, remaining_time3 = step(remaining_time=24,
                                                           schedule={'animatio
n': 3, 'Japanese': 3},
                                                           topic = "Japanese",
                                                           action= 7)
    call3 = [remaining_time3, reward3, schedule3]

    new_state4, reward4, schedule4, remaining_time4 = step(remaining_time=24,
                                                           schedule={'animatio
n': 3, 'physics': 3},
                                                           topic = "Japanese",
                                                           action= 7)
    call4 = [remaining_time4, reward4, schedule4]
    if reward3 >= reward4:
        coherent_more += 1

print("Coherent Reward >= Noncoherent Reward is True", coherent_more / len(np.
arange(10000)) *100, "% of the time, because of undercommitment penalties.")
```

```
Coherent Reward >= Noncoherent Reward is True 89.14999999999999 % of the tim
e, because of undercommitment penalties.
```

Input 5: step(remaining_time=24, schedule={'animation': 3, 'physics': 3}, topic = "animation", action= 1)

- Resampling the same topic should NOT change the length of the schedule but should override the hour allocation

```
In [11]: new_state5, reward5, schedule5, topic5 = step(remaining_time=24,
                                              schedule={'animation': 3, 'physi
cs': 3},
                                              topic = "animation",
                                              action= 1)
print("Resampling a topic overwrites hour allocation/action:\n", new_state5, r
eward5, schedule5, topic5)
```

```
Resampling a topic overwrites hour allocation/action:
 [23, 2] 0 {'animation': 1, 'physics': 3} graphic design
```

Test with random actions and topics

In [12]:
```
rewards = []
next_states = []
actions = []
schedule_lens = []
remaining_times = []

for episode in np.arange(10):
    print("--------------")
    print("\033[1;43m  EPISODE {} \033[0m".format(episode))
#     print("--------------")
    remaining_time = 24
    schedule = {}
    topic = np.random.choice(possible_topics)
    action = np.random.choice([0,1,3,5])

    ep_actions = []
    ep_sch_lens =[]
    while True:
        if action != 7:
            next_state, reward, schedule, topic = step(remaining_time, schedul
e, topic, action)
            next_states.append(next_state)
        elif action == 7:
            next_state, reward, schedule, remaining_time = step(remaining_time
, schedule, topic, action)
            next_states.append(next_state)

        if next_state == 0:

            actions.append(ep_actions)
            schedule_lens.append(len_schedule)
            remaining_times.append(remaining_time)
            break

        remaining_time, len_schedule = next_state

        action = np.random.choice(possible_actions)
        ep_actions.append(action)
        print('New STATE:', next_state)
        print("New ACTION:", action)
    print("Schedule:", schedule)
    print('Remaining Time:', remaining_time, "hrs")
    print("Reward:", reward)
    rewards.append(reward)
    sys.stdout.flush()
```

```
--------------
  EPISODE 0
New STATE: [24, 0]
New ACTION: 3
New STATE: [21, 1]
New ACTION: 1
New STATE: [20, 2]
New ACTION: 1
New STATE: [19, 3]
New ACTION: 7
Schedule: {'physics': 3, 'graphic design': 1, 'psych/linguistics': 1, 'Orgs':
2, 'LabStuff': 2}
Remaining Time: 15 hrs
Reward: 1.95
--------------
  EPISODE 1
New STATE: [24, 0]
New ACTION: 3
New STATE: [21, 1]
New ACTION: 1
New STATE: [20, 2]
New ACTION: 3
New STATE: [17, 3]
New ACTION: 5
New STATE: [12, 4]
New ACTION: 7
Schedule: {'graphic design': 3, 'video-editing': 1, 'animation': 3, 'R/statis
tical packages': 5, 'Orgs': 2, 'LabStuff': 1}
Remaining Time: 9 hrs
Reward: 2.6512500000000006
--------------
  EPISODE 2
New STATE: [21, 1]
New ACTION: 3
New STATE: [18, 2]
New ACTION: 5
New STATE: [13, 3]
New ACTION: 0
New STATE: [13, 3]
New ACTION: 0
New STATE: [13, 3]
New ACTION: 1
New STATE: [12, 4]
New ACTION: 5
New STATE: [7, 4]
New ACTION: 1
New STATE: [6, 5]
New ACTION: 7
Schedule: {'topology': 3, 'psych/linguistics': 3, 'Japanese': 5, 'graphic des
ign': 1, 'webtooning': 1, 'Orgs': 4, 'LabStuff': 1}
Remaining Time: 1 hrs
Reward: -1
--------------
  EPISODE 3
New STATE: [21, 1]
New ACTION: 7
Schedule: {'R/statistical packages': 3, 'Orgs': 1, 'LabStuff': 1}
```

```
Remaining Time: 19 hrs
Reward: -0.5
--------------
  EPISODE 4
New STATE: [21, 1]
New ACTION: 7
Schedule: {'topology': 3, 'Orgs': 4, 'LabStuff': 1}
Remaining Time: 16 hrs
Reward: 0.9
--------------
  EPISODE 5
New STATE: [19, 1]
New ACTION: 3
New STATE: [16, 2]
New ACTION: 0
New STATE: [16, 2]
New ACTION: 3
New STATE: [13, 3]
New ACTION: 5
New STATE: [8, 4]
New ACTION: 7
Schedule: {'video-editing': 5, 'topology': 3, 'psych/linguistics': 3, 'front-
end coding': 5, 'Orgs': 2, 'LabStuff': 2}
Remaining Time: 4 hrs
Reward: -1
--------------
  EPISODE 6
New STATE: [23, 1]
New ACTION: 5
New STATE: [18, 2]
New ACTION: 5
New STATE: [13, 3]
New ACTION: 3
New STATE: [10, 4]
New ACTION: 5
Schedule: {'front-end coding': 1, 'psych/linguistics': 5, 'topology': 5, 'vid
eo-editing': 3}
Remaining Time: 10 hrs
Reward: -1
--------------
  EPISODE 7
New STATE: [24, 0]
New ACTION: 0
New STATE: [24, 0]
New ACTION: 5
New STATE: [19, 1]
New ACTION: 1
New STATE: [18, 2]
New ACTION: 1
New STATE: [17, 3]
New ACTION: 3
New STATE: [14, 4]
New ACTION: 1
New STATE: [13, 4]
New ACTION: 3
New STATE: [10, 4]
New ACTION: 7
```

```
Schedule: {'video-editing': 5, 'front-end coding': 1, 'Japanese': 3, 'topolog
y': 3, 'Orgs': 2, 'LabStuff': 1}
Remaining Time: 7 hrs
Reward: 3.225
--------------
```

EPISODE 8

```
New STATE: [24, 0]
New ACTION: 3
New STATE: [21, 1]
New ACTION: 0
New STATE: [21, 1]
New ACTION: 0
New STATE: [21, 1]
New ACTION: 1
New STATE: [20, 2]
New ACTION: 1
New STATE: [19, 3]
New ACTION: 1
New STATE: [18, 4]
New ACTION: 1
New STATE: [17, 5]
New ACTION: 1
New STATE: [16, 5]
New ACTION: 3
New STATE: [13, 5]
New ACTION: 5
New STATE: [8, 6]
New ACTION: 3
Schedule: {'physics': 3, 'Japanese': 3, 'R/statistical packages': 3, 'animati
on': 1, 'video-editing': 1, 'psych/linguistics': 5}
Remaining Time: 8 hrs
Reward: -1
--------------
```

EPISODE 9

```
New STATE: [19, 1]
New ACTION: 3
New STATE: [16, 2]
New ACTION: 5
New STATE: [11, 3]
New ACTION: 0
New STATE: [11, 3]
New ACTION: 0
New STATE: [11, 3]
New ACTION: 7
Schedule: {'animation': 5, 'webtooning': 3, 'Japanese': 5, 'Orgs': 3, 'LabStu
ff': 1}
Remaining Time: 7 hrs
Reward: 2.559375
```

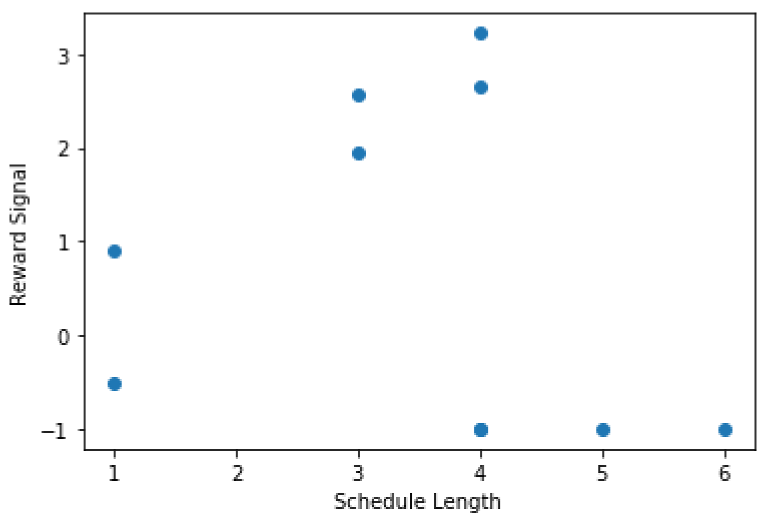
```
In [13]: print("rewards:", rewards)
         print("actions:", actions)
         print("schedule lengths:", schedule_lens)
```

```
rewards: [1.95, 2.6512500000000006, -1, -0.5, 0.9, -1, -1, 3.225, -1, 2.55937
5]
actions: [[3, 1, 1, 7], [3, 1, 3, 5, 7], [3, 5, 0, 0, 1, 5, 1, 7], [7], [7],
[3, 0, 3, 5, 7], [5, 5, 3, 5], [0, 5, 1, 1, 3, 1, 3, 7], [3, 0, 0, 1, 1, 1,
1, 1, 3, 5, 3], [3, 5, 0, 0, 7]]
schedule lengths: [3, 4, 5, 1, 1, 4, 4, 4, 6, 3]
```

Plot Schedule Length vs. Reward

- the longer the schedule, the more likely karoshi happens, so reward should go down as length of schedule increases
- length of schedule doesn't necessarily mean lots of hours are allocated per reward, so karoshi might not happen, BUT longer schedules increase chances of incoherence, so reward also diminished

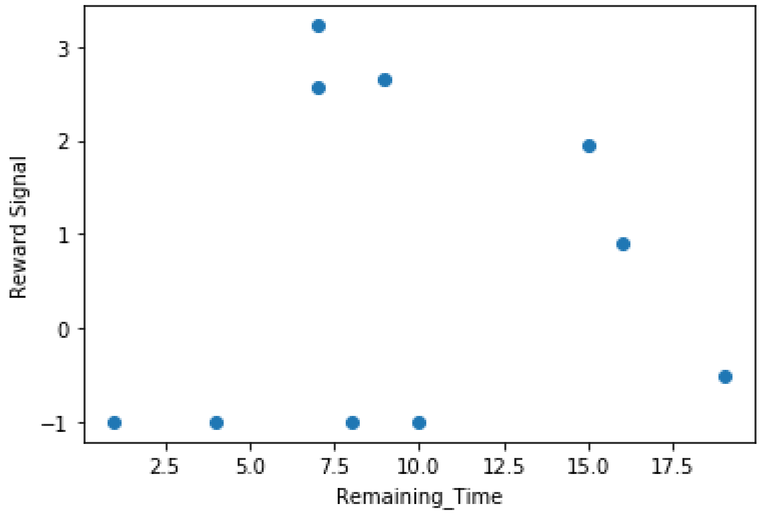
```
In [14]: plt.plot(schedule_lens, rewards, "o")
         plt.xlabel("Schedule Length")
         plt.ylabel("Reward Signal")
         plt.show()
```



Plot Remaining Times vs. Reward

- When remaining time is under 6-8, the reward should reapidly decrease,since karoshi happens due to lack of sleep/ bonuses are not rewarded for being sleep-deprived.
- When remaining time is 24, meaning the schedule is empty, undercommitment is not penalized but also not rewarded, leaving the agent with a reward of 0.
- The sweet spot is to have around 8-15 hours of remaining time, which accounts for sleep and free time. Having more free time than that makes the agent unlikely to be able to accrue schedule-fulfillment points.
-

In [15]:
```
plt.plot(remaining_times, rewards, "o")
plt.xlabel("Remaining_Time")
plt.ylabel("Reward Signal")
plt.show()
```



In [ ]: