In [1]:
```
import numpy as np
import pandas as pd
import warnings
from datetime import datetime, timedelta
from matplotlib import pyplot as plt


warnings.filterwarnings('ignore')
```

Import Data

In [2]:
```
telescopes = ['12-meter','alma','apex','aste','iram','jcmt','lmt','sma','smt','spt']
```

In [3]:
```
starttime = datetime(2019,10,3,6)
endtime = datetime(2019,10,14,0) # not included
timestamps = np.arange(starttime, endtime,
                       timedelta(hours=6)).astype(datetime)
databook = {}
for ts in telescopes:
    databook[ts] = dict.fromkeys(timestamps)
```

In [4]:
```
for ts in telescopes:
    for t in timestamps:
        filepath = "data/"+ ts +"/"+ t.strftime("%Y%m%d_%H:%M:%S")
        try:
            df = pd.read_csv(filepath, delim_whitespace=True, skiprows = 1, header =
            df.columns = ["date", "tau225", "Tb[k]", "pwv[mm]", "lwp[kg*m^-2]","iwp[k
            df['date'] = pd.to_datetime(df['date'], format = "%Y%m%d_%H:%M:%S")
            databook[ts][t] = df
        except FileNotFoundError:
            databook[ts][t] = None
# databook is a dictionary of dictionaries of dataframes
# keys: telescope names
# values: dictionaries of dataframes for one telescope
# databook[telescope_name] is a dictionary of dataframes for one telescope
# keys: timestamps when the forecast is made
# values: forecast dataframe (None if missing)
```

Baseline Model

For the baseline, we do not take any uncertainty into account. We only use the latest prediction for each time.

Since tau225 has a negative relationship with the photo quality, we use -tau225 here to calculate the reward based on the following steps.

1. For each telescope, calculate their reward for the day according to their schedule.

According to the scheduling file that EHT has sent to us, we calculate the reward for each telescope only based on the following schedule provided by EHT for Tue 24 Apr 2018 (whether the telescopes will be triggered all the time as the schedule needs further confirmation)

Station	Obs. start time (UTC)	Obs end time(UTC)	Total GBytes
ALMA	03:02:00	13:09:00	22830.7

Station	Obs. start time (UTC)	Obs end time(UTC)	Total GBytes
APEX	03:02:00	15:11:00	26153.8
PICOVEL	03:02:00	07:25:00	8800.0
SPT	03:02:00	15:00:00	26953.8
LMT	05:53:00	15:45:00	22215.3
SMTO	07:22:00	15:45:00	18030.7
JCMT	09:42:00	15:45:00	12123.0
SMAP	09:42:00	15:45:00	12123.0

Due to the property of our data, we approximate the time to o'clock as following:

Station	Obs. start time (UTC)	Obs end time(UTC)	Total GBytes
ALMA	03:00:00	13:00:00	22830.7
APEX	03:00:00	15:00:00	26153.8
PICOVEL	03:00:00	08:00:00	8800.0
SPT	03:00:00	15:00:00	26953.8
LMT	06:00:00	16:00:00	22215.3
SMTO	08:00:00	16:00:00	18030.7
JCMT	10:00:00	16:00:00	12123.0
SMAP	10:00:00	16:00:00	12123.0

In [5]:
```
def day_reward(telescope_name, day_current_str, end_day_str, start_time, end_time, us
    # return all the weather prediction data we can get before day_current for the pe

    split_day_current = day_current_str.split('-')
    split_day_end = end_day_str.split('-') # include this day

    day_current = datetime(int(split_day_current[0]),int(split_day_current[1]),int(sp
    day_end = datetime(int(split_day_end[0]),int(split_day_end[1]),int(split_day_end[

    if not use_as_evaluate:
        mask = [t < day_current for t in databook[telescope_name]]
        t_valid = np.array([t for t in databook[telescope_name]])[mask]

        df_all = pd.concat([databook[telescope_name][t] for t in t_valid], axis =0)
    else:
        df_all = pd.concat([databook[telescope_name][t] for t in databook[telescope_r

    df_tau_all = df_all.groupby('date').agg({'tau225':lambda x: list(x)}).reset_index

    df_tau_all['latest'] = df_tau_all['tau225'].apply(lambda x: x[-1]) # baseline onl


    df_tau_all = df_tau_all[(df_tau_all.date >= day_current) & (df_tau_all.date < day


    # calculate the reward for each day based on the schedule
    df_tau_all['day'] = df_tau_all.date.apply(lambda x: str(x).split(' ')[0])
    df_tau_all['time'] = df_tau_all.date.apply(lambda x: int(str(x).split(' ')[1][0:2

    df_tau_all = df_tau_all[(df_tau_all.time >= int(start_time)) & (df_tau_all.time <
    df_tau_day = pd.DataFrame(-df_tau_all.groupby('day')['latest'].mean())

    return df_tau_day
```

2. Weighted sum the reward for each telescope according to the total Gbytes. (so far we have not taken the telescopes '12-meter','aste','iram' into account as we haven't found corresponding schedule and weights)

In [6]:
```
weight_telescope = [0, 22830.7, 26153.8, 0, 0, 12123.0, 22215.3, 12123.0, 18030.7, 26
schedule_telescope = [[0,1], [3,13], [3,15], [0,1], [0,1], [10,16], [6,16], [10,16],

dict_schedule = dict(zip(telescopes, schedule_telescope))
dict_weight = dict(zip(telescopes, weight_telescope))
```

In [7]:
```
def all_day_reward(day_current_str, end_day_str):
    telescopes_day_reward = day_reward(telescopes[0], day_current_str, end_day_str, d
    for i in telescopes[1:]:
        telescopes_day_reward += day_reward(i, day_current_str, end_day_str, dict_sch
    return telescopes_day_reward
```

3. Choose the optimal N days and also return whether triggering today or not

In [8]:
```
# for future uncertainty, revise this function to return more information
def decision_making(day_current_str, end_day_str, days_to_trigger):
    # day_current_str: YYYY-MM-DD (str) (included)
    # end_day_str: YYYY-MM-DD (str) (included)
    # days_to_trigger: days to trigger (int)
    days_to_trigger = all_day_reward(day_current_str, end_day_str).sort_values(by='la
    if day_current_str in days_to_trigger:
        print('We suggest triggering on today')
    else:
        print('We DO NOT suggest triggering on today')
    print('And we suggest to trigger by the following sequence: {}'.format(np.array(s
```

So far we do not automatically count down the remaining days because we want to keep enough flexibility for EHT as the remaining days might not follow what we suggest as the real application might have unexpected conditions.

4. Model Evaluation

In [9]:
```
def best_path_afterwards(start_day_str, end_day_str, days_to_trigger, days_have_trigg
    # start_day_str: YYYY-MM-DD (str) (included)
    # end_day_str: YYYY-MM-DD (str) (included)
    # days_to_trigger: days to trigger (int)
    # days_have_triggered: days acutally triggered (list of str)
    telescopes_day_reward = day_reward(telescopes[0], start_day_str, end_day_str, dic
    for i in telescopes[1:]:
        telescopes_day_reward += day_reward(i, start_day_str, end_day_str, dict_sched

    all_path = telescopes_day_reward.sort_values(by='latest', ascending = False)
    best_path = all_path[:days_to_trigger]
    print('The best path to trigger based on ground-truth is {}'.format(np.array(sort
    print('The total reward based on best path is {}'.format(best_path['latest'].sum(
    if days_have_triggered is not None:
        print('The total reward based on real path is {}'.format(all_path.loc[days_ha
    return all_path
```

5. Use baseline model in a Case (choose 5 days from 10 days between 10.5 ~ 10.14)

On day1 (10.05): `Trigger`

In [10]:
```
decision_making('2019-10-05', '2019-10-14', 5)
```

```
We suggest triggering on today
And we suggest to trigger by the following sequence: ['2019-10-05' '2019-10-06' '20
19-10-07' '2019-10-10' '2019-10-14']
```

On day2 (10.06): Not Trigger

In [11]:
```
decision_making('2019-10-06', '2019-10-14', 4)
```

```
We DO NOT suggest triggering on today
And we suggest to trigger by the following sequence: ['2019-10-07' '2019-10-09' '20
19-10-11' '2019-10-13']
```

On day3 (10.07): `Trigger`

In [12]:
```
decision_making('2019-10-07', '2019-10-14', 4)
```

```
We suggest triggering on today
And we suggest to trigger by the following sequence: ['2019-10-07' '2019-10-08' '20
19-10-09' '2019-10-11']
```

On day4 (10.08): Not Trigger

In [13]:
```
decision_making('2019-10-08', '2019-10-14', 3)
```

```
We DO NOT suggest triggering on today
And we suggest to trigger by the following sequence: ['2019-10-09' '2019-10-13' '20
19-10-14']
```

On day5 (10.09): `Trigger`

In [14]:
```
decision_making('2019-10-09', '2019-10-14', 3)
```

```
We suggest triggering on today
And we suggest to trigger by the following sequence: ['2019-10-09' '2019-10-13' '20
19-10-14']
```

On day6 (10.10): Not Trigger

In [15]:
```
decision_making('2019-10-10', '2019-10-14', 2)
```

```
We DO NOT suggest triggering on today
And we suggest to trigger by the following sequence: ['2019-10-13' '2019-10-14']
```

On day7 (10.11): Not Trigger

In [16]:
```
decision_making('2019-10-11', '2019-10-14', 2)
```

```
We DO NOT suggest triggering on today
And we suggest to trigger by the following sequence: ['2019-10-13' '2019-10-14']
```

On day8 (10.12): `Trigger`

In [17]:
```
decision_making('2019-10-12', '2019-10-14', 2)
```

```
We suggest triggering on today
And we suggest to trigger by the following sequence: ['2019-10-12' '2019-10-13']
```

On day9 (10.13): `Trigger`

In [18]:
```
decision_making('2019-10-13', '2019-10-14', 1)
```

```
We suggest triggering on today
And we suggest to trigger by the following sequence: ['2019-10-13']
```

On day10 (10.14): Have no days to trigger

In conclusion, the real-path we suggest to trigger is:

In [19]:
```
real_path = ['2019-10-05','2019-10-07','2019-10-09','2019-10-12','2019-10-13']
```

Model Evaluation

In [20]:
```
all_state = best_path_afterwards('2019-10-05', '2019-10-14', 5, days_have_triggered =
```

```
The best path to trigger based on ground-truth is ['2019-10-05' '2019-10-06' '2019-
10-09' '2019-10-11' '2019-10-13']
The total reward based on best path is -54244.05535937293
The total reward based on real path is -4260443.875346138
```