Python code description of the Feature Based Scheduler

Elahesadat Naghib

Key terms and notations

t	GMT time in Julian Date,
$\tau_s(t)$	beginning of the night that t lies in,
$\tau_e(t)$	end of the night that t lies in,
$id(t)$	ID of the field that is visited at t, $id \in \{1, 2, 3, ..., 4206\}$,
$\theta_l(i,t)$	time of the last visit of field i before $\tau_s(t)$, $\theta_l(i,t) = \infty$ if field i is not visited before $\tau_s(t)$,
$\theta_l^N(i,t)$	time of the last visit of field i between $\tau_s(t)$ and t, $\theta_l^N(i,t) = \infty$ if field i is not visited in this interval,
$n(i,t)$	number of the visits of field i before $\tau_s(t)$
$n^N(i,t)$	number of the visits of field i between $\tau_s(t)$ and t, $0 \leq n^N(i,t) \leq 3$
$SM_{i,j}$	slew time from field i to field j, $SM \in \mathbb{R}^{4206\times4206}$ is a given slew matrix,
$alt(i,t)$	altitude of the center of field i at t, $-\frac{\pi}{2} \leq alt(i,t) \leq \frac{\pi}{2}$,
$ha(i,t)$	hour angle of the center of field i at t, $-12 \leq ha(i,t) \leq 12$
$\tau_{rise}(i,t)$	rising time of field i above the 1.4 airmass horizon at current night, $\tau_{rise}(i,t) = -\infty$ if i never sets down,
$\tau_{set}(i,t)$	setting time of field i below the 1.4 airmass horizon at current night, $\tau_{set}(i,t) = \infty$ if i never sets down,
$M_\phi(t)$	percent of the Moon's surface illuminated at t,
$M_{sep}(i,t)$	Moon's separation from field i at t, $0 \leq M_{sep}(i,t) \leq \pi$,
$W_1,\ W_2$	given constant time window in which a revisit is valid, $0 < W_1 < W_2$
timeslots	
visibility	
brightness	

1. FB SCHEDULER CODE IN SUMMARY

Code repository on github: FB Scheduler

Main files:

- **CreatDB.py**: creates the database, tables, data structure, and feeds the model parameters into the database.
- **FieldDataGenerator.py**: evaluates fields' predictable data such as altitude, at certain time intervals and writes them on the database.
- **FBDE.py**: is where the heart of the scheduler is. it (1) reads the data in, stores the field data into field objects, (2) for each visit, loops over the field objects and (3) update the timing and the fields for the next visit decision.
- **UpdateDB.py**: reads the out put of FBDE.py (the visit sequence), (1) evaluates the statistics and history dependent variable required for the next episode scheduling and (2) writes them into the database

2. FB SCHEDULER CODE IN DETAILS

Class DataFeed
- connect to database
- read in ID, RA, Dec, Science label, $\theta_l(i,t)$, and $n(i,t)$ of all fields
- read in $alt(i,t)$, $ha(i,t)$, visibility, cloud coverage, and brightness of the fields for all time intervals
- read in model parameters: ∞, ϵ, Exposure time, W_1, W_2, maximum number of visit per night
- read in slew times from file
- create and initialize field(*FiledState*) objects, one object for each field
- create episode(*EpisodeStatus*) object, one object for one episode of scheduling (mostly a night)

Class Scheduler(DataFeed)
- scheduler
  - initialize episode
  - create output sequence structure
  - *while* $t <$ end of the episode
    - for all fields
      - update field feasibility
      - evaluate cost of feasible fields
    - make the decision of next visit based on the costs
    - simulate the visit {evaluate the visit time, update the target field object}
    - record visit in a text file and in the output array
    - update the episode {update t, update current field, update current filter}
    - update all fields variable{$alt(i,t)$, $ha(i,t)$, visibility, cloud coverage, and brightness, $SM_{i,currentfield}$}
  - save the output array containing sequence of the visits for the episode

Class EpisodeStatus

`keeps track of the timing and other changing variables with time`
- initial variables: {$\tau_s(t)$, $\tau_e(t)$, time intervals where the fields data are calculated at}
- updatable variables: {t, decision number, last visited field, current filter}

Class FieldState

`keeps track of the state of each field`
- initial variables for the i^{th} object (a field): {ID, RA, Dec, Science label, $\theta_l(i,t)$, and $n(i,t)$}
- data stored in the $i'th$ object (a field):
- {$alt(i,t)$, $ha(i,t)$, visibility, cloud coverage, and brightness, for all time slots, and $SM_{i,allotherfields}$}
- updatable variables:
- {$alt(i,t)$, $ha(i,t)$, visibility, cloud coverage, and brightness, for the current time, $SM_{i,currentfield}$}
- updatable variables by calculation: {time since last visit, time to become invisible, feasibility, cost}

3. RUN THE SCHEDULER

3.1 Required packages

Code is developed in Python2.7.10, and the following packages are required to be installed:

- PyEphem
- Numpy
- SQLite3
- JSON
- Pandas
- time
- Matplotlib
- ProgressBar

3.2 Required data

Labeled field information: "/NightDataInLIS/Constants/fieldID.lis", can be downloaded here
Slew matrix: "/NightDataInLIS/Constants/slewMatrix.dat.lis", can be downloaded here

3.3 Quick start with FB scheduler code

Scheduling of the LSST is a history dependent procedure, and the validity of the decisions in a certain night depends on the successful scheduling of all previous nights, evaluating the history dependent data (such as total number of visits), and storing them in the database.

1. **init_setup.py** is a one-time procedure: creates a database in the main directory with field's data for 10 nights, starting from 2021/01/01, (takes around 3 minutes)
2. **run.py**: schedules 10 nights starting from 2021/01/01. It also updates the database after each night of scheduling is completed. log and numpy output of individual nights will be store in "/Output" directory, and the mp4 output of each night will be stored in "/Visualizations" directory (all takes around 10 minutes).