SCOPES

System for **C**oordinating **O**bservational **P**lanning and **E**fficient **S**cheduling

Nicolas Unger

May 30, 2025

v0.4.x

Contents

1 Introduction

SCOPES is a Python package designed to automate and optimize the scheduling of astronomical observations for ground-based telescopes with dedicated instruments like cameras or spectrographs. It addresses the challenge of allocating shared telescope time among various observational programs, each with its unique scientific objectives and priorities, into a single night by optimizing several criteria and constraints. By maximizing the use of available time and adhering to observational and scientific constraints, SCOPES ensures that the operation of the telescope is both effective and efficient.

In this context, a "schedule" refers to a planned sequence of targets, typically stars or any object of interest, that the telescope is set to observe for a specified duration before moving on to the next target. This schedule can cover an entire night of observation or just parts of the night. SCOPES only builds schedules for a single night, it does not do any multi-night or long-term planning. Any long-term planning can be done as a pre-processing and then giving to SCOPES for a specific night the list of targets that result from your long term planning. For example, similarity is done in [Handley et al., 2024b] in the context of Doppler programs.

SCOPES is based on the scheduling framework presented by [van Rooyen et al., 2018], designed to optimize observation schedules by calculating a score for each observation at a specific time using a customizable rank function that adheres to principles of fairness, sensibility and efficiency (more on this in Sect. 3). The schedule is then built from a list of candidate observations, respecting all constraints and instructions set by the program coordinators, with the framework simultaneously optimizing the order of observations to minimize overhead time between switches and maximize the overall score of the schedule.

This tool is intended to be used by telescope managers and administrators who have access to all the programs and targets to be observed with the telescope. It can also be used by individual observers, but the set-up required might be too demanding if it is only going to be used in a few isolated nights. The idea is that the setup is done once, including the ability to have to update the input observations only and run the scheduler every night.

Custom GPT to interact with this documentation

You can interact with this documentation with ChatGPT via the custom GPT found here: `https://chatgpt.com/g/g-uEcapCq94-scopes`.

It contains this documentation and can help you understand how SCOPES works by asking questions and having a conversation about any topic covered in this document.

2 Installation

To install SCOPES, open your terminal and run the following command:

```
pip install scopes-astro
```

You can then import the package into your Python code with:

```
import scopes
```

Notes:

Along with `scopes`, it will also install the following dependencies if they are not already installed:

- `numpy`
- `pandas`
- `matplotlib`
- `astropy`
- `astroplan`
- `tqdm`
- `pytz`
- `timezonefinder`

Additionally, if you want to use the interactive plotting functionalities you will also need `plotly` installed.

The minimum Python version required is 3.8.

3 Scheduling Strategy

Following the framework outlined by [van Rooyen et al., 2018], an effective scheduling system for astronomical observations must incorporate the principles of fairness, sensibility, and efficiency:

1. **Fairness** ensures equitable distribution of observing time across various observational programs. This principle also respects the scientific priorities for each program.

2. **Sensibility** assesses a target's suitability for observation within predefined observational constraints. This principle incorporates veto constraints to confirm that only targets meeting the necessary observational limits, or sensible observations, are considered.

3. **Efficiency** optimizes the scheduling of observations to identify the best moments for specific targets, thereby maximizing telescope usage and data quality.

We then encapsulate these three principles into a single *rank* function:

$$R_n = f(n) \cdot \prod_{x=1}^{X} \upsilon_x(n) \cdot \frac{\sum_{y=1}^{Y} \varepsilon_y(n)}{Y} \qquad \forall n = 1, ..., N \tag{1}$$

where n is an individual observation, $f(n)$ is a measure of fairness, $\upsilon(n)$ is a measure of sensibility via boolean veto functions, and $\varepsilon(n)$ is a measure of efficiency.

Each part is built with individual merit functions which define the specific constraints that an observation has to comply for it to be considered in the schedule. These merit functions are then combined in the rank function according to their classification. Merits are all normalized to be between 0 and somewhere close to 1, to make sure that a low priority but high value merit doesn't win to a high priority but low value merit.

Sensibility merits (or veto merits) are combined by taking their product. Veto merits are boolean functions that decide if a target can be observed or not, and thus any hard limit that is not respected would make the entire rank function equal zero and thus

is enough to remove the target from consideration. Even though in general these are boolean functions, intermediate values between 0 and 1 are allowed. This would happen in a situation where a certain merit is not a hard limit, but has a transition from 0 to 1. In which case any value between 0 and 1 means that the situation to observe this target is not ideal, but if for example the efficiency or fairness merits are high enough then this target might still be selected for observation.

Efficiency merits are combined as an average. This is because these merits are merely optimizations and do not invalidate an observation from happening if they are zero. Because each target can have any number Y of efficiency merits, we take the mean of all the efficiency merits to not bias in favor of targets with more merits.

Fairness merits are combined in product (similar to veto merits) with the difference that these merits should always hover around 1, with slightly higher values than 1 if the target has a higher priority and vice versa if the target has a lower priority. It's a way of slightly pushing its score higher or lower depending on its priority. It can be thought of as a percentage increase of the score. If the fairness merits come out as 1.1, this means that this target will have a 10% higher score than normal because of its higher priority. As we'll see in Sect. 4.1, depending on the chosen fairness merits, this can happen either based on intrinsic scientific priority or because the program that this targets belongs to has been under observed and thus receives a slight push to balance the time allocation.

This rank function is evaluated for each observation at any moment in time to obtain a single score, which is then used to rank observations to aid in scheduling. The specific scheduling algorithm will be described later in Sect. 6.

4 Merits

Now we'll go more into detail of the specific merit functions that are available in SCOPES. These are just the merit functions that are already implemented. The main idea of this framework is that any type of merit that works on any available parameter can be created and added to the rank function. These can easily be custom defined by the user (see Sect. 4.4).

4.1 Fairness merits

Fairness merits mainly measure the scientific priorities of targets and programs, making sure that the allocation of targets and the shared time of the telescope is "fair". Here by fair we mean that the allocated time for each program is respected, that the priorities of programs and targets are taken into account, and also that bad weather events affect all programs equitably. These merits are combined in product:

$$f(n) = \prod_i p_i \tag{2}$$

where p_i are the individual fairness merits.

Let's outline the specific fairness merits that are implemented in SCOPES and how they work.

4.1.1 Priority

The priority merit respects the scientific priorities that programs and targets were assigned to. This is a simple integer priority scale which contains four levels of priority:

- **0**: Top priority. The priority level should be used sparingly.
- **1**: High priority
- **2**: Normal priority
- **3**: Low priority

These levels of priority can be assigned to both programs and individual targets. And the idea is that the priority of a target is only relative to the priority of its program, and not a global priority. Meaning that just because a target has high priority doesn't mean that it will be more likely to be observed than a low priority target for a higher priority program.

To achieve this constraint and also to map these priority values to a value close to 1, I use the following system. Then both the program and target priorities are combined to generate a single Combined Priority (CP), that is then used to compare all targets with each other.

First, we map the priority levels described above to a more usable number. Both for programs and targets, there is a base (B) and offset (O) value. Then for a priority $p \in [0, 3]$, the priority value is calculated as:

$$V = B + O \cdot (2 - p) \tag{3}$$

this applies to both programs and target, so we will have a Program Value (PV) and a Target Value (TV).

The idea is then that the Combined Priority (CP) is obtained by adding PV and TV:

$$CP = PV + TV \tag{4}$$

Where TV can be thought of itself as an offset to PV which is its base value. Here lies the concept of coupling the target priority as a relative value to its program priority.

The default values in SCOPES are:

$$\begin{aligned} PB &= 1 \\ PO &= 0.1 \\ TB &= 0 \\ TO &= 0.05 \end{aligned}$$

The bases for the program and target values should be 1 and 0, respectively. However, the offsets can be modified, as these indicate how much each priority level represents in the rank function.

Let's see an example:

Assuming the Base and Offset values presented above, for a program with priority 1 and a target within it with priority 3:

- Program Value (Eq. 3) $= 1 + 0.1 \cdot (2 - 1) = 1.1$.
- Target Value (Eq. 3) $= 0 + 0.05 \cdot (2 - 3) = -0.05$.
- Combined Priority (Eq. 4) $= 1.2 - 0.05 = 1.05$.

Which can be thought of as this target having a 5% boost to its rank score due to its individual and program priority.

4.1.2 TimeShare

This merit makes sure to respect the time-share that each program was allocated to. This can be done over any time frame. Usually, programs get assigned how much time they can use over a semester. This time is sometimes expressed in number of nights or hours, but for SCOPES this time has to be expressed as a percentage of the total time.

The main variable that this merit function uses is the percent difference ($\Delta\%$) between the allocated time and the actual used time of a program. The telescope administrator has to keep track of how much time was used by each program and indicate this to SCOPES to calculate $\Delta\%$.

It uses a modified sigmoid function to calculate the merit. The specific shape can be set with the parameters α, β, and γ. The exact formula is:

$$m = \frac{\gamma}{\left(1 + \exp\left(\left(\frac{\Delta\%}{\beta}\right)^{\alpha}\right)\right)} + \left(1 - \frac{\gamma}{2}\right) \tag{5}$$

The shape of this function can be seen in Fig. 1. It hovers at a merit value of 1, forming a flat stretch near $\Delta\% = 0$. This plateau persists until a specific threshold, where the merit increases for negative $\Delta\%$ and decreases for positive $\Delta\%$. Following this divergence, the function levels out again into a plateau, maintaining a steady merit for $\Delta\%$ beyond.

The function's design allows a permissiveness or tolerance to how much time a program can use relative to its allocated time. This built-in tolerance lets a program run slightly over or under its allocated time before its merit is adjusted. The parameters involved, α, β, and γ, fine-tune this behavior: α shapes the steepness of the transition, β sets the transition's midpoint between stable and variable merit, and γ decides the magnitude of merit change when a program's actual time use is significantly above or below the allocated one (γ can be thought of as a percentage of reward or penalization that a program receives when it is considerably above or below its allocated time, respectively).

NOTES:

- α has to be a positive **odd** integer for the sigmoid function to behave correctly.
- The default values in SCOPES are $\alpha = 3$, $\beta = 5$, and $\gamma = 0.05$.
- These default values ensure that the time-share is balanced over a time frame of around a week, with a tolerance of 5% (defined by β).
- γ should not be too high, as you don't want a too strong incentive or penalization of individual programs over a single night. This can cause the schedule to only include targets from one program.

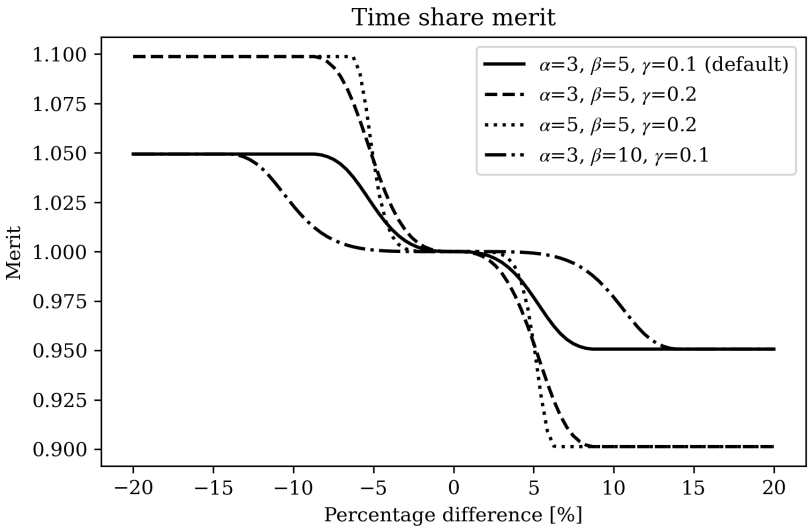

Figure 1: Time share merit based on percent difference between actual used time and allocated time for an observing program. The three parameters that define the shape of the merit are α, a measure of how sharp the transition is from 1 to the cap level, β, a measure of the tolerance for over or under observation, and δ, the limit of how much the merit goes above and below 1.

- This merit is evaluated once at the beginning of the night, and the values for each program are fixed for the scheduled night. That's why the balancing of time happens over a sequence of several nights by slightly, on average, increasing the number of targets for the under observed programs and decreasing the targets for over observed programs.

4.2 Veto merits

Veto merits, or sensibility merits, are those related to the observability of a target. Meaning, they make sure that it make sense to observe the target at a particular time. Some usual limitations are that the target should be high enough in the sky, respect telescope hardware limits, not observe during the day, etc. These are called veto merits, because any one of these merits by themselves can make an observation unavailable to the scheduler if it evaluates to zero.

Veto merits are combined in product to make sure that any merit that evaluates to 0 automatically sets the rank score of the entire observation to 0:

$$V(n) = \prod_{x=1}^{X} \upsilon_x(n) \tag{6}$$

where x is an index for each veto merit and υ are the merits themselves.

4.2.1 Airmass

This merit sets the desired limit in airmass for the target. It uses a hyperbolic tangent function centered at the desired airmass limit (a_0) and a measure of how much tolerance there is around this limit with the parameter α. The exact formula is:

$$m = \frac{\tanh\left(\frac{(a_0 - a)}{\alpha}\right) + 1}{2} \tag{7}$$

If α is very small, this function behaves similarly to a boolean step function at the desired airmass limit. The function is shown in Fig. 2 with different values of α.

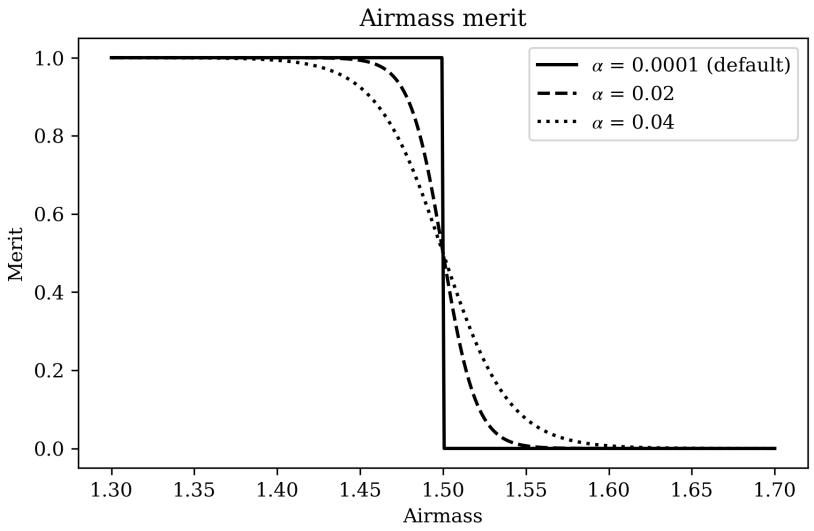

Figure 2: Airmass merit function. The parameter α determines the steepness or tolerance around the desired airmass limit (which in this case has been set to 1.5 as an example).

4.2.2 Altitude

This merit sets the desired limits in altitude of the target. Typically, these are set due to telescope hardware limits. For example, in some telescope the tracking has issues if a target goes through the zenith, and thus an altitude limit of just below 90 degrees can be set to avoid this scenario.

$$m = \begin{cases} 1 & \text{if alt} > \text{alt}_{\text{min}} \text{ and alt} < \text{alt}_{\text{max}} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

In SCOPES, the default values for alt_{min} and alt_{max} are 20 and 90 degrees, respectively.

4.2.3 AtNight

This merit ensures that the produced schedule is limited to the night. In this case no argument is needed as this is set when creating the Night object where the desired observational night time limits have to be indicated. That is which twilight to start and end observations on (civil, nautical or astronomical).

4.2.4 MoonSeparation

This merit sets separation limits between the moon and the target.

$$m = \begin{cases} 0 & \text{if } \theta < \theta_{\text{lim}} \\ \left(\frac{\theta - \theta_{\text{lim}}}{\theta_{\text{start}} - \theta_{\text{lim}}}\right)^{\alpha} & \text{if } \theta_{\text{lim}} < \theta < \theta_{\text{start}} \\ 1 & \text{if } \theta \geq \theta_{\text{start}} \end{cases} \tag{9}$$

There is the limit separation (θ_{lim}) below which this merit acts as a veto, and a starting separation (θ_{start}) above which the merit will evaluate to 1. These parameters can vary depending on the target or observing wavelength, for example. Additionally, the α parameters sets the steepness of the merit between θ_{lim} and θ_{start}. If a simple step function is desired, then one can simply set $\theta_{\text{lim}} = \theta_{\text{start}}$[1]. In fig. 3 the merit function is displayed with different parameter values and $\theta_{\text{start}} = 30°$.

NOTE:

- Calculating the separation to the moon for a target is something that can be calculated in advance before the scheduling. The moon only moves close to 5 degrees across the background stars during a ∼9 night. This means that within that error one can pre-calculate this assuming a fixed position of the moon. Considering the actual path of the moon can also be done in advance and filter out the targets that come close to the moon during the night. This can save significant computation time to the scheduler.

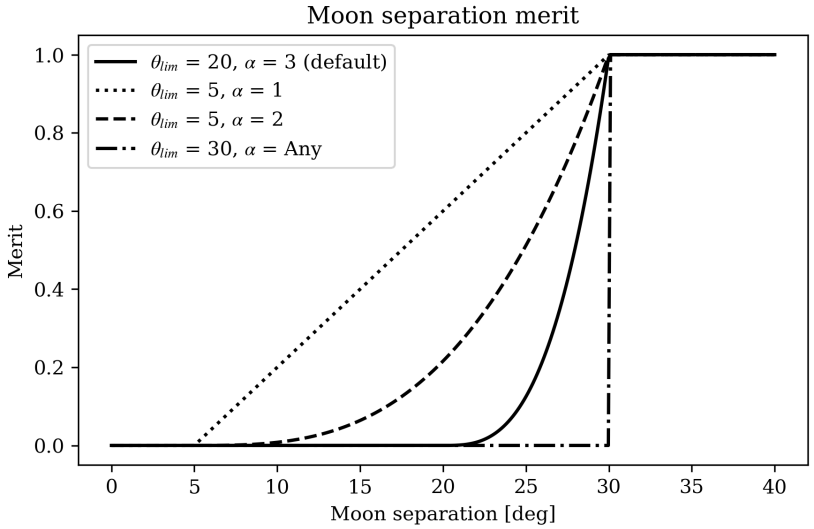

Figure 3: Moon separation merit function. The starting and limit separations can be defined as well as α which is the steepness parameter.

4.2.5 StartTime

It's a simple veto merit that checks that the current time is not before the specified time. This merit is used when an observation should only be scheduled after a specific time.

[1]This won't create a problem because in that case the middle part of Eq. 9 will never be evaluated as θ cannot be strictly between θ_{lim} and θ_{start} if $\theta_{\text{lim}} = \theta_{\text{start}}$.

4.2.6 EndTime

It's a simple veto merit that checks that the observation doesn't go over a specified time. The scheduler uses this merit when a global end time for the schedule is set. In that case, this merit gets added to all observations to prevent the schedule to past that time. The user can still assign this merit to individual targets if they find it useful.

4.3 Efficiency merits

Efficiency merits are not mandatory, meaning that any one of them could evaluate to zero, but it doesn't make the observation impossible. These are more related to the specific timing of an observation. Either given in the instructions by the PI (if any), as well as optimizing the conditions of the observation like observing a target as high as possible in the sky to get the best quality data.

4.3.1 CulminationEfficiency

This merit optimizes for observing targets "high" in the sky, but it doesn't only do that. If one created a merit that is simply proportional to its airmass or relative altitude to culmination, then targets are only going to be observed when they are that, close to their meridian passing. Doing this presents a problem for targets that are setting or rising, that is targets that never reach their culmination during the current night, yet are still perfectly observable for a short time either at the beginning or at the end of the night.

We do not want to ignore those targets because ideally one wants to observe a target repeatedly on the largest time window of a year as possible while it's observable from your location. This means observing it during the few weeks when the target is rising and only visible for a few hours at the end of the night, and conversely, for the few weeks when the target is setting and only visible for a few hours at the beginning of the night.

To take these targets into account in the scheduling, SCOPES uses a custom culmination efficiency merit that maps an extended nighttime window onto the actual nighttime window to determine when a particular target peaks in this merit. This extended range starts at the earliest culmination point of any star still observable at the night's start and ends at the latest culmination point of any star still observable at the night's end.

When a target is evaluated in this merit, first the actual culmination time of the target is determined (if it is within this extended time window), which is then mapped onto the actual nighttime range where observations will be taken (typically within nautical or astronomical twilights). It's a simple one-to-one mapping between two different time ranges. That new mapped time is when that star will peak in this merit function.

The method shifts observation priorities throughout the night—prioritizing setting stars early on, stars near their culmination around the middle, and rising stars towards the end of the night. This method is a compromise between observing stars at their highest point, but also giving priority to stars that are setting or rising.

NOTE: This merit should not be used if it conflicts with another merit that prefers specific observing times, like time critical or phase specific observations. This merit is meant for targets where the specific timing during the night is not as important. Where generally only a good and high enough in the sky observation is important.

[The actual merit function is a bit complicated and can't be shown in a simple analytical form. Well.. its analytic form is actually just a Gaussian centered at this mapped time described above, and with a standard deviation of 4 hours. I know that setting the same

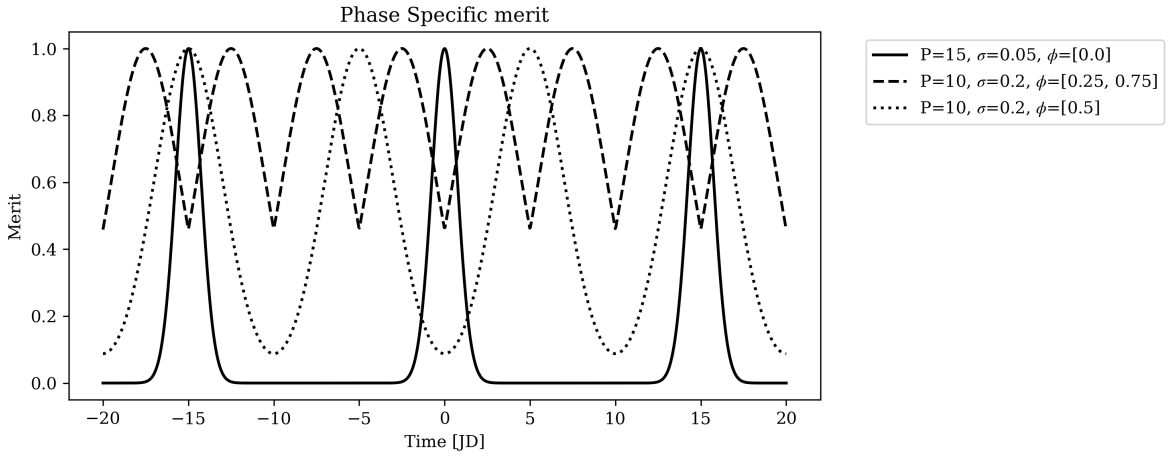

Figure 4: Graphic representation of the phase specific merit. Here, all curves have a fixed epoch of 0.

width to all targets is not ideal, as the effective "width" of the skypath of a target can change. I've been meaning to find a better suited analytical form of this merit that makes physical sense while still maintaining this mapped time feature.]

4.3.2 PhaseSpecific

This merit is used when the observation should be taken at specific phases of a periodic time interval. In exoplanet detections, for example, this is used when observations have to be taken at a specific phase of the orbit of a planet. Its analytic expression is a slight modification of the merit presented by [Granzer, 2004]:

$$m(x) = \sum_{i=-1}^{1} \exp\left(-\frac{1}{2}\left(\frac{x(t) - (\phi - i)}{\sigma}\right)^2\right) \tag{10}$$

$$x(t) = \frac{\mathrm{mod}\,(t - t_0, P)}{P} \ , \tag{11}$$

where t is time, P is the period, ϕ is the desired phase at which to observe, and σ is the standard deviation of the Gaussian in phase space. If more than one phase is required, then the merit will be taken as the maximum of the merits calculated for each phase:

$$m = \max_{\phi}\ m(x(t), \phi) \ . \tag{12}$$

This is done instead of summing the contributions of each phase to avoid biasing towards higher merits as a consequence of having several phases. A graphic representation of this merit can be seen in Fig. 4.

If you use this merit, the priority of the target should be set to high or top to make sure that it is selected at the correct time. Otherwise, some other target that might peak in its culmination efficiency merit just around that time might win over this one. But because the phase specific instruction is a sort of "time critical" observation, it should be given priority whenever it is the correct time to observe it.

The analytic expression of Eq. 10 ensures that σ is in units of phase and behaves as a normal standard deviation (as long as it's small enough). Because the equation is

effectively adding the contribution of three separate Gaussians, if σ becomes large enough, these start to fuse and the actual shape of each Gaussian starts to be affected by the other two, which "enlarges" them. This starts to happen at around a value of $\sigma = 0.4$.

Another thing to take into account with this merit is that, for example, if one wants an observation to be taken at a phase of $\phi = 0.25$ with a tolerance of ± 0.05, then for this to happen in practice, the value of σ has to be set higher than the actual width of the tolerance zone. This is because in a Gaussian with its peak normalized to 1, the value at 1σ and -1σ is close to 0.6. A merit of 0.6 in the rank function will score low compared to other observations, and would thus almost surely not be selected. To remedy this, σ should be set to a value around twice as large, $\sigma = 0.1$ in this case, at which the merit at $\phi = 0.2$ (and $\phi = 0.3$) would evaluate to around 0.88 where there can still be a chance for it to be selected.

This ad-hoc enlargement of σ is only heuristic in nature and needs to be derived empirically to what works for your use case.

Usually, the periods that are considered when using this merits is of several days, meaning that this merit can span several nights. Which means that this merit will only "activate" (have a value close to 1) at a specific night and time at which it is ideal for that observation to be taken.

4.3.3 AirmassEfficiency

This merit function is meant as an efficiency merit but based on the airmass value of the target. It is not advised to use this merit in the general scheduling phase because it will highly favor target that are high in the sky and will penalize target that never reach high altitudes during the night. That's why it's preferable to use the CulminationEfficiency merit (see Sect. 4.3.1) in those cases.

This merit is used internally in SCOPES for the optimization phase of the scheduling. Once a proposed schedule is found, it can be improved by permuting observations and thus ensuring that the targets are as high as possible in the sky as they can be, given the current schedule. To do this, the CulminationEfficiency merit is removed (if any) and this ArimassEfficiency merit is added. The optimization algorithm is performed over this new set of merits, obtaining an optimized version of the proposed schedule.

The exact formula is:

$$m = \frac{1}{a} \tag{13}$$

4.4 Custom Merits

SCOPES allows the user to define any custom merit. The only constraint is that it has to work on one of the variables available in the `Observation` class (see Sect. 5.5 (although this could be adapted as well by adding custom attributes to `Observation` for use in the merits).

To define a custom merit function that can be used in SCOPES it has to follow these requirements:

- The first argument has to be called `observation` which is an instance of `Observation` (explained in detail in Sect. 5.5).

- Do any calculation you want using the available attributes of `Observation` and return a float that is ideally limited to the range $[0, 1]$. Where 1 means the current situation of the observation is most favorable for this merit and 0 means it's the least favorable.
- When building your custom merit, think of which type it will be (fairness, veto, efficiency) as this will be required later, but also to keep in mind how it will interact with all the other merits.

How to use your custom merit in SCOPES will be explained in Sections 5.3 and 5.4.

5 Scheduling Components

In this section we will explain the different components of SCOPES (python objects), their purpose and how to set them up. These are:

- `Night`
- `Program`
- `Merit`
- `Target`
- `Observation`
- `Overheads`
- `Plan`

These can be imported into python from the `scopes.scheduling_components` module. For example:

```
from scopes.scheduling_components import Night
```

5.1 `Night`

The `Night` object contains information about the specific night for which a schedule is to be created. To initialize it, one only needs to give three parameters:

- `night_date` (type: `datetime.date`): This is always the date at which sunset for that night happens in local time of where the telescope is located.
- `observations_within` (type: `str`): Between which twilights observations should happen. The options are: `civil`, `nautical`, and `astronomical`. These are then used for the AtNight merit (Sect. 4.2.3).
- `observer` (type: `astroplan.Observer`): Indicated the geographical location of the telescope using the `Observer` type from the `astroplan` package [Morris et al., 2018].

When the `Night` object is initialized, it internally calculates many relevant quantities (such as the exact timing of the different twilights) that are important for the actual scheduling at a later stage.

5.2 Program

The `Program` class defines an individual observing program of the telescope. Observing programs typically are run by one or only a handful of researchers with a unique scientific goal or objective. This is increasingly the case in bigger telescopes that are shared between many scientific objectives or even institutes. In practice, there can be dozens of different observing programs sharing the time of only one telescope. Hence, the scheduling challenge is to create a nightly observing schedule that follows the instructions and constraints of all programs equitably.

If your telescope only has one program, then you can just define one simple program and continue reading, as some of the information in this section is only relevant if many programs are to be included.

To initialize a `Program` instance, the following parameters have to be given:

- `progID` (type: `str`): A string representing the program ID or name.
- `priority` (type: `int`): A integer that indicates the priority level of this program. See Sect. 4.1.1 for an explanation of the scale that needs to be used to set this priority value.
- `time_share_allocated` (type: `float`): The percentage of the total available telescope time that is awarded to this program.
- (optional) `plot_color` (type: `str`): Optional parameter to indicate the HEX code of the color with which this program will be plotted.

Internally, attributes called `time_share_current` and `time_share_pct_diff` are created. If the TimeShare merit is going to be used (Sect. 4.1.2) then it is very important to update the `time_share_current` attribute before running the schedule for the night by using the `set_current_time_usage()` method. In this method one simply indicates the percentage of the total time that this program actually used until now, and internally it will also update `time_share_pct_diff` which tracks the percent difference between the allocated time and the currently used time.

How the total time, or from which point this metric should be counted on, is up to the user. Many observatories do this on a semester timeframe. Each semester, scientific and political decisions are made on how much time each program gets to use. Thus, the user will have to keep track of how much time was actually used by each program in order to be able to use the TimeShare merit effectively.

5.3 Merit

The `Merit` class defines and evaluates individual merit functions. Given a particular merit function, which can be any of the already defined merits from SCOPES (the ones presented in Sect. 4.1 through 4.3) or any custom created merit function (Sect. 4.4), then a `Merit` instance has to be created for it. This object will store the name and function itself, as well as the type of merit and the specific parameters that it will use:

- `name` (type: `str`): The name for this merit.
- `func` (type: `Callable`): The function where the merit is defined (use any from the `scopes.merits` module or your custom defined merit function).

- `merit_type` (type: `str`): Which type of merit it is. The options are: [fairness, veto, efficiency]. This will indicate in which section of the rank function (Eq. 1) it will be used.
- `parameters` (type: `Dict[str, Any]`): A dictionary that contains the parameters' name and value pairs to indicate the specific parameters to be used in this instance of this merit.
- `weight` (type: `float`, default: `1.0`): The weight assigned to this merit. This weight acts as a multiplicative factor on the output of the merit function, effectively adjusting its influence in the total rank computation. A higher weight increases the importance of this merit relative to others of the same type.

For example, if you use the airmass merit, the type has to be set to `"veto"` and the `parameters` could be set to {`"limit": 1.5`} to set an upper limit of 1.5 for the airmass. Then if for another target the airmass limit can be relaxed, you would have to create a new `Merit` instance with the same `name`, `func` and `merit_type`, and set `parameters` to {`"limit": 1.7, "alpha": 0.1`}, where the limit is now increased to 1.7 and with $\alpha = 0.1$ the transition from a merit of 1 to 0 is smoother (see Sect. 4.2.1). If you also want this particular airmass condition to have a stronger veto power compared to others, you could set `weight` to a higher value such as `2.0`.

5.4 `Target`

With the `Target` class, one defines each individual celestial object that will be considered for observation. To initialize an instance of `Target` the parameters are:

- `name` (type: `str`): The name of the target.
- `program` (type: `Program`): The program that this target is part of.
- `coords` (type: `astropy.coordinates.SkyCoord`): A `SkyCoord` object with the coordinates of the target.
- `priority` (type: `int`): The priority level for this target. See Sect. 4.1.1 on how this priority should be set.
- (optional) `comment` (type: `str`): An optional comment for this target that will be displayed in the final schedule for the human observer to see.

Once the `Target` instance has been created, one has to add the corresponding merits that should be followed for it. This can be done with the `add_merit()` method with an instance of `Merit` as the input. Merits can also be added in bulk with the `add_merits()` method, where several merits can be added in bulk with a list of `Merit` instances.

5.5 `Observation`

The `Observation` class is the main part of SCOPES. It manages the calculations of all merits and the rank function to keep track of the score of each observation. To do this, it calculates the position of the target on the sky at any given time and keeps track of its position in the schedule. To initialize an instance of `Observation`, the only parameters needed are:

- `target` (type: `Target`): The target for this observation.
- `duration` (type: `float`): The duration of this observation expressed in seconds.
- `instrument` (type: `str`): The name of the instrument that this program uses. This is relevant when a telescope has more than one instrument available and can switch between them.

There are many internal methods in the `Observation` class, but the user will not have to interact with them, as these are all used internally by the scheduler. Refer to the code documentation to see what they are.

5.6 `Overheads`

The `Overheads` class is to define how much time will pass when transitioning from one observation to the next. For example, the one overhead that basically all telescopes will have is the slew time. This is the time the telescope needs to physically move to point from one location is the sky to another. The specific will depend on the configuration of each telescope, but this will typically consist of movement in two axes, altitude and azimuth. Even in telescopes on an equatorial mount where azimuth is not the direction in which the telescope moves, they typically still have a dome around them that does move have to move in azimuth.

Other possible overheads of your telescope could include: readout of the CCD camera, focusing, pointing, changing filters, etc. The only default type of overhead in `Overheads` is the slew time, which has to be defined at initialization:

- `slew_rate_az` (type: `float`): The slew rate of the telescope/dome in the azimuth direction in degrees per second.
- `slew_rate_alt` (type: `float`): The slew rate of the telescope in the altitude direction in degrees per second.
- (Optional) `cable_wrap_angle` (type: `float`): The azimuth angle where the cable wrap limit is at, in degrees.

Note: SCOPES assumes that the azimuth and altitude movements of the telescope happen simultaneously. This means that when calculating the total slew time, it takes whichever slew direction takes longer.

Many telescopes have cable wrap limits. These are typically an angle in azimuth that the rotation of the telescope can't go through because the telescope is not able to rotate freely. The cables connecting the instruments eventually run out of length, and thus the telescope can't continue rotating in the same direction and must go the other way to avoid breaking any cables. If this is the case in your telescope, you can indicate the azimuth angle that the telescope shouldn't cross, and this will be taken into account when calculating the path the telescope takes when transitioning from one target to the next.

Once the `Overheads` object is created, one can add any number of additional custom overheads. To do this, you have to create a function that takes two consecutive observations as input and returns a time as the number of seconds of that overhead. These arguments have to be called `observation1` and `observation2`, respectively.

For example, there could be a CCD readout overhead that depends on the instrument being used. So in that case the overhead doesn't really depend on what observation comes next, so the function would only check which instrument is being used in the current observation and returns the number of seconds that the CCD readout takes for that instrument. On the other hand, you could add an overhead for the change of instruments, where it checks if the instrument will change from the current observation to the next and return how much time it takes for your telescope to change instruments.

Any attribute available in `Observation`, `Target`, or `Program` can be used by accessing the nested properties of each. You can also create new attributes not available by default in SCOPES. For example, if the filters of an observation are relevant, and you want to add an overhead for the change of a filter, you can simply add a `filter` attribute to your `Observation` instances and then access these in your custom overhead functions.

To add a custom overhead function to the `Overheads` you call the `.add_overhead()` method with the following inputs:

- `overhead_func` (type: `Callable`): Your custom overhead function.
- `can_overlap_with_slew` (type: `bool`): A boolean indicating if this overhead overlaps with the slew of the telescope. By default, it's `False`. If `True`, `Overheads` will take the time of whichever takes longer.

Being accurate the overheads of your telescope is important for SCOPES to work correctly. If you under- or overestimate the overheads, this can lead to serious mismatches between the schedules created by SCOPES and what can actually be done in your telescope. Any error on the overheads means that this error accumulates over time, with the potential of either not having enough time to execute the schedule or having extra time that SCOPES couldn't take into account. As well as potential mistakes in the exact timing of time sensitive observations. So make sure to create estimate the overheads of your telescope as closely as possible to the real ones.

5.7 `Plan`

The `Plan` class is the class to build schedule itself. In essence, it's just an ordered list of observations which precise timings for when each starts and stops. As the user, you will not have to create the `Plan` itself, as this is done by the schedulers. You will only have to interact with the final finished `Plan` instance that the scheduler outputs. With a `Plan` you can view the actual observations in table form or create various plots to visualize it. To simply view a plan one can call `print()` on it for a tabular form of the observations, which program/instrument they are part of, the start and durations, as well as any comment that was left by the program PI's.

The available methods for the user are:

- `.print_stats()`: Prints several key information about the plan like its length (number of observations), the score, total observation time, total overhead time, their ratios and average airmass.
- `.plot(display=True, path=None)`: This will create an altitude/airmass vs time plot where each observation will be shown as a segment of the targets' altitude as a function of time and at which point of the night it was scheduled to be observed. The optional parameters are: `display` if you want the plot to be shown, and `path` if it should be saved by indicating the path as `path/to/file/filename.extension`.

- `.plot_interactive(path=None)`: The same as `.plot()` but in interactive form using the `plotly` package.
- `.plot_polar(display=True, path=None)`: A plot in polar coordinates of altitude and azimuth showing the path that the telescope will take during the night. This type of plot is useful to identify if the telescope is taking any inefficient routes or if close to 180-degree rotation are happening, which can add unnecessary overheads.
- `.plot_altaz(display=True, path=None)`: It creates an altitude vs time and azimuth vs time plots. It shades the overheads to give a better feel of how long each overhead is, and it shades the background in colors according to the instrument of the observation to better visualize when there are instrument changes. This plot is also useful to identify if the schedule is efficient and is not doing any overly inefficient movements.
- `.to_df()`: Creates and returns a `pandas.DataFrame` of the schedule. This is what is printed in text form when calling `print()` on the `Plan`.
- `.to_csv(*args, **kwargs)`: Saves a CSV file of the resulting DataFrame. Calling this method is equivalent to calling `.to_df()` and then calling the native pandas' method `.to_csv()`. It takes any argument that pandas' `.to_csv()` can take.

6 Schedulers

In this section I will describe the `scheduler` module, that is the part of SCOPES that actually takes your observations and puts them in an efficient and optimized sequence for the night.

A base `Scheduler` class is defined that includes useful methods related to the scheduling of observations. However, the actual scheduling is done with separate classes that inherit from this base `Scheduler` class. The current implementation of SCOPES provides the `generateQ` class for quick and efficienct scheduling of observations.

6.1 The `generateQ` scheduler

First the an instance of the `generateQ` scheduler is created. The input for the initialization is the `Night` for which the scheudle will be created, the list of `Observation`s to be considered, and the `Overheads` object where all the telescope overheads have been defined. Optionally it is also possible to set start or end times for the schedule, this is used when only a part of the night should be scheduled.

The initializaiton will do some basic checks, assign the `Night` to all the provided observations, and do some preliminary calculations for each like calculating the path on the sky that they will take during the night. Depending on the number of observation that are given in the input, this step might take some time, between a few seconds up to a few minutes. In practice it doesn't help having more than 100-150 observations as consideration for a schedule. Depending on the duration of the observations, a normal night might only have between 20-50 observations, so adding several hunderds of observation will make the whole night more inefficient. Try filtering out some observations before this step if you have too many.

How it works

Once the `generateQ` instance has been created, the scheduling is run by calling the `.run()` method. One the optional parameters is `K`, which will be explained shortly, and the other is `max_plan_lenght` which allows you to indicate how many observations you want to be planned. By default its `None` which means that it will fill the available time, which is either the entire night or the time limits indicated when initializing the `generateQ` instance. But maybe you just want to schedule the next 2 or 3 observations in which case that parameter becomes useful.

The scheduling starts by creating an empty `Plan` instance which will be filled with observations until its reacheas the required time or length. Then, the first phase of the scheduling process involves forward planning, where the scheduler evaluates a set of top `K` observations based their rank score at the start time of the Plan. For each of those `K` observations, the `forwardP` method fills the next 5 observations based on a naive greedy search which simply takes the top ranking observation at each step and appends it to a temporary Plan. The observation among the `K` with the highest Plan score at the end of this simulation is selected as the first observation in the actual Plan.

Once an observation is selected, the scheduler applies a Local Search Heuristic (LSH) optimization. This step involves reordering the selected observations to minimize overheads such as telescope movement and instrument changes. This LSH algorithm is based on the algorithm presented by [Handley et al., 2024a] in Appendix C. In short, what it does is take a random observation of the current Plan, and then it places this observation at all possible positions with insert operations and reevaluate the overall Plan score. If it found a new position that improves the overall Plan score it moves it to that position. Then a new random observation is taken (without replacement) and again looking if there is a better position in the Plan for this observation that increases the overall Plan score. This process is repeated for all observation in the Plan. Because this is just one iteration of this process and the order in which the observation are selected can alter the final resulting Plan, the entire optimization is repeated as many times as necessary until the optimization no longer finds a better Plan. This usually takes only 3-5 iterations of the LSH optimization. This ensures that we are packing the observations as tightly as possible to minize overheads at every step.

At this stage there is only one observation in the plan, so this LSH optimization will not do anything at this stage.

The iterative selection process then repeats, where the scheduler again considers the next top `K` observations sorted by their rank score if it is places right after the first observation. It repeats the `forwardP` to add the next 5 observation, and selects the observation with the highest overall plan score. Then the LSH optimization to minimize overheads is run again.

This process continues, adding one observation at a time to the schedule until the maximum plan length is reached the available time has been filled.

After constructing a full plan, the scheduler performs a final LSH optimization to further refine the schedule. Instead of minimizing overheads, this optimization step aims to maximize the overall plan score by reordering observations to achieve better observing conditions, such as a higher average airmass, which contributes to the quality of the data collected.

This is a very simple sequential scheduler which works in giving a reasonably good schedule optimized for both small overheads and data quality. Its simplicity is good by providing a schedule in a relatively short time. Its runtime with 100-150 observation usually takes less than 3-5 minutes.

Better algorithms are surely possible and I encourage anyone willing to try their hand at writing a better implementation by implementing their own custom scheduler.

I have experimented with other type of alogrithms such as the `DPPlanner` and `BeamSearchPlanner` schedulers defined in the `scopes.scheduler` module that perform the scheduling using dynamic programming and beam search techniques, respectively. However, these have not been optimized and are quite slow and produce worse schedules than `generateQ`.

6.2 Custom schedulers

As with the merits and overheads, SCOPES allows the user to define their own custom schedulers. This is done by creating a new class that inherits from the base `Scheduler` class to initialize the scheduler. Then typically a `.run()` method is defined that will do the actual scheduling.

7 Usage

In the GitHub repository there is Jupyter notebook that details and shows how to use SCOPES from a simple example to a full night of observations. It can be found here:

`https://github.com/nicochunger/SCOPES/blob/main/scopes_example.ipynb`

8 Frequently Asked Questions

1. **How can I prioritize certain targets over others in my observation schedule?**

   - You can prioritize targets by assigning them a priority level (0: Top priority, 1: High, 2: Normal, 3: Low) within their respective programs. SCOPES uses a combined priority system that considers both program and target priorities to influence the scheduling. See Sect. 4.1.1 for more information.

2. **What should I do if my observing program has used more or less time than allocated?**

   - Use the `TimeShare` merit, which accounts for the percent difference between allocated and used time, to adjust the schedule accordingly. This merit only works when used over more than one night. It will balance time distribution over several nights to ensure fair time allocation among programs. See Sect. 4.1.2 for more information.

3. **How can I ensure that my observations are scheduled only during the night?**

- SCOPES provides an `AtNight` veto merit that ensures observations are only scheduled during nighttime as defined by civil, nautical, or astronomical twilight limits.

4. **What is the difference between veto merits and efficiency merits?**

   - Veto merits (sensibility merits) are constraints that must be met for a target to be considered for observation; if any veto merit evaluates to zero, the observation is discarded. Efficiency merits optimize the timing of observations but do not prevent them if they evaluate to zero.

5. **Can I define my custom merit functions in SCOPES?**

   - Yes, SCOPES allows you to create custom merits by defining functions that operate on available attributes in the `Observation` class. These custom merits can be integrated into the scheduling process alongside pre-defined merits.

6. **How can I manage telescope overheads like slew time or instrument changes?**

   - SCOPES includes an `Overheads` class where you can define telescope-specific overheads, including slew rates for azimuth and altitude. You can also add custom overhead functions if your telescope has unique requirements.

7. **What should I do if my target needs to be observed at a specific phase of one of its planet's orbit?**

   - You can use the `PhaseSpecific` efficiency merit, which allows scheduling based on the phase of a periodic event, such as a planet's orbit. Adjust the merit's parameters to match the desired observational phase. See Sect. **??** for more information.

8. **Can I schedule observations that must be completed before a specific time?**

   - Yes, SCOPES offers an `EndTime` veto merit, which ensures that observations do not extend beyond a specified time limit. This is useful for ensuring that time-sensitive observations are completed within a required timeframe.

9. **How can I visualize the final schedule generated by SCOPES?**

   - After the schedule is created, it will return an instance the `Plan` class which has several methods to visualize the schedule in various formats, including altitude vs. time plots, polar plots, and azimuth plots. These visualizations help assess the efficiency of the schedule.

10. **What if I need to create a custom scheduler for my unique requirements?**

    - SCOPES allows the creation of custom schedulers by extending the base `Scheduler` class. You can implement your own scheduling logic and integrate it with SCOPES's framework to meet specific needs.

References

[Granzer, 2004] Granzer, T. (2004). What makes an automated telescope robotic? *Astronomische Nachrichten*, 325(6):513–518.

[Handley et al., 2024a] Handley, L. B., Petigura, E. A., and Mišić, V. V. (2024a). Solving the Traveling Telescope Problem with Mixed-integer Linear Programming. , 167(1):33.

[Handley et al., 2024b] Handley, L. B., Petigura, E. A., Mišić, V. V., Lubin, J., and Isaacson, H. (2024b). Automated Scheduling of Doppler Exoplanet Observations at Keck Observatory. , 167(3):122.

[Morris et al., 2018] Morris, B. M., Tollerud, E., Sipőcz, B., Deil, C., Douglas, S. T., Berlanga Medina, J., Vyhmeister, K., Smith, T. R., Littlefair, S., Price-Whelan, A. M., Gee, W. T., and Jeschke, E. (2018). astroplan: An Open Source Observation Planning Package in Python. , 155(3):128.

[van Rooyen et al., 2018] van Rooyen, R., Maartens, D. S., and Martinez, P. (2018). Autonomous observation scheduling in astronomy. In *Observatory Operations: Strategies, Processes, and Systems VII*, volume 10704 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, page 1070410.