Automated Scheduling for the Orbiting Carbon Observatory 3 Mission

Alan Moy, Amruta Yelamanchili, Steve Chien, Annmarie Eldering, Ryan Pavlick
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
{firstname.lastname}@jpl.caltech.edu

Abstract

We describe the automated scheduling system in development and in use for the Orbiting Carbon Observatory-3 Mission (OCO-3), which launched to the International Space Station in May 2019. We first describe the high level scheduling problem of scheduling the four types of observations: nadir, glint, target, and snapshot area map. We then describe the major complexity of OCO-3 scheduling - enforcing geometric visibility constraints for snapshot area map and target modes. We also describe an automated scheduling of instrument pointing calibration. We then describe current and related work as well as future directions for the scheduling of OCO-3.

Introduction

The Orbiting Carbon Observatory 3 (OCO-3) is a NASA (National Aeronautics and Space Administration) instrument for measuring atmospheric CO_2. It is mounted on the International Space Station (ISS) on the Japanese Experiment Module - Exposed Facility (JEM-EF). OCO-3 will enable identification of CO_2 sources and sinks and study of changes in CO_2 levels over time, both throughout the day and across seasonal cycles. It is made up of three high-resolution spectrometers integrated into one structure with a common telescope. It identifies CO_2 indirectly by measuring the intensity of solar radiation reflected off of CO_2 molecules in the air column. OCO-3 was installed on the ISS in May 2019, and has a planned operational life of three years (NASA 2019).

OCO-3 has four (4) operational modes:

- Nadir mode - observations directly below the instrument, suitable for measurement over land
- Glint mode - observations taken over the ocean that point near the glint spot (point of maximum solar reflection) to counter the reduced reflection of the ocean
- Snapshot Area Map mode for rectangular regions of interest
- Target mode for points of interest

The instrument is outfitted with an agile two-dimensional pointing mirror, known as the Pointing Mirror Assembly (PMA). The PMA allows rapid transitions between modes, as well as multiple scans over the snapshot area map and target mode targets in a single overflight.

OCO-3 is mounted on an area of the ISS where the rotating solar panels of the spacecraft regularly enter its field of view. The light reflected off of the solar panels could damage the instrument, so flight software prevents the instrument from pointing in this unsafe region. However, we do not ever want to schedule observations in this unsafe region, so this poses an additional challenge of checking the visibility of targets during scheduling.

The core of the scheduling system is an adaptation of the Compressed Large-scale Activity Scheduler Planner (CLASP) (Knight and Chien 2006; Rabideau et al. 2010). CLASP supports scheduling of mapping (areal coverage) observations respecting geometric constraints such as instrument visibility and illumination conditions. CLASP will be used nominally to generate a schedule lasting for two weeks, that will be uploaded to the instrument weekly.

For the PMA calibration routine we were required to schedule observations over a fixed grid of azimuth and elevation points in the instrument's frame of reference in order to calibrate the pointing accuracy of the PMA. CLASP only deals with targets as fixed areas in a geographic coordinate system. For this, we used some of the geometric reasoning available in CLASP, but created a separate scheduler for this problem.

In the remainder of this paper we describe the design of the scheduling system that tries to maximize the utility of the science data OCO-3 acquires while respecting illumination, time, visibility, and instrument safety constraints. We also describe the scheduler developed for the PMA calibration routine.

Prototypical CLASP Scheduling Problem

In the basic areal coverage scheduling problem, the inputs are a set of science campaigns, each of which contains:

- target regions of interest on the Earth's surface
- illumination constraints
- a priority

The goal is to produce an observation schedule to view these regions as many times as possible while respecting instrument constraints such as mode transition times and keepout

Copyright © 2019 California Institute of Technology. U.S. Government sponsorship acknowledged.

regions of the instrument field of view. Science campaigns can be target regions or single point locations. We generate a gridded approximation of target regions for faster computation, to get a set of target points.

CLASP uses the CSPICE Toolkit provided by the Navigation and Ancillary Facility (NAIF) (Acton 1996) at JPL to do geometric reasoning regarding the visibility swaths of instruments from the spacecraft they are attached to. The size, shape, and location of the swaths depend on the position and orientation of the spacecraft, and the field-of-view of the instrument. The planning horizon is broken up into fixed duration observations, and CLASP computes the intersection between the target grid points and the observations. Initially, a one-pass greedy scheduling algorithm was used to place observations according to the priority of the targets they cover.

Problem Statement

The original CLASP problem statement (Knight and Chien 2006):

Given:

- a set of regions of interest $R = \{r_1, ..., r_n\}$
- a temporal knowledge horizon (hst, het) over which we know the vehicle's activities
- a set of observation opportunities $O = \{o_1, ..., o_n\}$ within the horizon (hst, het) where each $o_i \in O$ consists of a start ($o.start$) and a duration ($o.duration$)
- a set of instrument swaths I $=\{i_1, ..., i_n\}$ where $\forall(o_i \in O)\exists(r_i, i_i) \mid (\text{grid}(o_i) \in \text{grid}(r_i)) \wedge (\text{grid}(o_i) \in \text{grid}(i_i))$
- a scoring function $U(r_i)$
- a bound on memory M_{max}
- a rate at which memory is used while the instrument is on $\dot{M}_{fill}$
- a rate at which memory is recovered during downlink $\dot{M}_{drain}$

Our goal is to select $A \subseteq O$ to maximize U(r_i) $\forall r_i \in R$ subject to instrument constraints.

For OCO-3, the data acquisition rates, downlink rate, and memory capacity allow the instrument to continuously take data over the regions of interest without violating memory constraints, so we do not model them in this adaptation for simplicity.

Operational Modes

There are four operational modes OCO-3 can be in: nadir mode, glint mode, snapshot area map mode, and target mode. For each mode, there is an associated set of science campaigns to be observed with that mode. If at any point the constraints are not met for any operational mode, the instrument enters standby mode and will not collect any science data until another observation is scheduled. Instrument calibration data can be taken during this time as well.

Nadir Mode

The simplest and lowest priority of the science campaigns is the nadir campaign, which is viewed by instrument in nadir mode. The nadir point is the point directly below the instrument on the Earth's surface. The nadir campaign consists of all landmass on the Earth and has a solar zenith angle (SZA) illumination constraint. When there are no other higher priority observation opportunities over land, and the illumination conditions are met at the nadir point, a nadir mode observation can be scheduled. While in nadir mode, the instrument is just pointed at the nadir spot.

Glint Mode

The glint campaign consists of all bodies of water on the Earth and has an equal priority to the nadir campaign. The instrument observes the glint campaign in glint mode. Instead of pointing at the nadir spot, OCO-3 will point off-nadir slightly in the direction of the glint spot. The glint spot is the location where solar radiation is maximally reflected off the Earth's surface toward the instrument. The angle between where the instrument is pointed and the glint spot is called the glint offset. This value is calculated as a function of the SZA at the glint spot. The glint campaign also has an SZA constraint, but at the glint spot rather than at the nadir spot.

The function for calculating the glint offset (g_o) is not a continuous function, but rather a piece-wise step function parameterized by the SZA at the glint spot (g_{sza}). The glint spot changes throughout time based on the spacecraft position and illumination conditions.

$$g_o = \begin{cases} offset_1 & 0 \leq g_{sza} \leq sza_1 \\ offset_2 & sza_1 < g_{sza} \leq sza_2 \\ \dots & \\ offset_n & sza_{n-1} < g_{sza} \leq sza_n \end{cases} \tag{1}$$

The scheduling software must determine when this value should change. It achieves this by sampling the glint spot and calculating the SZA at some interval. When g_o changes across an interval, additional samples are taken to determine more accurately when the g_o should change. Intervals for sampling must be chosen carefully. If the interval is too large, the glint offset may change twice within the interval, and the scheduler will not see the first change. To avoid this issue, the interval is chosen based off of the maximum rate of change of the SZA at the glint spot and the function from SZA to glint offset.

Snapshot Area Map Mode

OCO-3 has an area mapping mode dedicated to observing regions in key locations. Each region is approximately an 80 km square but is actually specified as a rectangle aligned along the spacecraft track with an across track and along track extent. In snapshot area map mode, successive scans are taken over the region during a single overflight to view all parts of it. The PMA moves to leading of the square and stays at that relative position. The forward motion of the ISS allows one stripe of data over the square to be viewed. When

Figure 1: An illustration of observations taken by OCO-3 from the ISS in snapshot area map mode (purple swaths) over the Los Angeles Basin (NASA 2019).

one stripe is done, the PMA moves across track in order to take the next stripe over the region. Figure 1 shows an illustration of an intermediate result of an observation taken in snapshot area map mode. The completed snapshot will provide dense carbon dioxide data over an area of about 250 square miles (NASA 2019).

Target Mode

Campaigns viewable by the target mode consist of single points on the Earth's surface. Many of the targets of this type are validation targets. These locations are part of the Total Carbon Column Observing Network (TCCON) (of Technology 2019), which are ground sensors that take independent CO_2 measures. By observing these sensors it's possible to compare OCO-3's measurements to the sensors' measurements to validate the instrument is working as expected. Observations taken in target mode are similar to those taken in area mapping mode, but rather than covering a larger region with multiple non-overlapping stripes in a single overflight, the same stripe is covered repeatedly.

We try to schedule exactly two target mode observations each day. CLASP does not have built-in support for this use case, but we can work around that issue by taking advantage of its extensible design. This is discussed in a later section.

Target mode and snapshot area mapping mode both have higher priority than nadir and glint mode. We discuss the interaction between the two modes later. In both, the scheduling tool takes as input a list of targets and tries to schedule short observations of those targets. This differs from glint and nadir modes, in which the instrument will continuously collect data as long as the constraints are met. We elaborate on the kinds of targets relevant to each mode in later sections.

Scheduling Target Mode Observations

The original version of CLASP did not have any support for being able to schedule a maximum number of observations of a specific instrument mode. For mission operations, it was deemed necessary to observe two validation targets per day. It is possible to have less than two possible observations of validation targets due to illumination conditions. But for days when there are more than two observations for validation targets, we want only two to be scheduled.

We can limit the number of validation target observations to two per day by introducing a new resource, which is a built-in CLASP feature. This resource is filled with two units at the beginning of each day in the planning horizon. Each time a validation target is scheduled, this resource is decremented by one. No target mode observations can be scheduled on a given day if there is no available resource for it to use.

Since no other observation mode requires a specific number of observations be scheduled, we can now ensure we schedule two validation target observations each day by giving the target campaign to have the highest priority. This way, whenever there is a validation target observation opportunity, it will be taken, up to the limit.

While giving validation targets the highest priority works in allowing us to schedule two per day whenever possible, it may be problematic when there are more than two observation opportunities in a single day. It is possible an observation in target mode may preclude an opportunity for an observation in snapshot area map mode. When that's the case, some solutions will be better than others. Snapshot area map observations give the highest science return, so we want to schedule as many as possible. If there are some validation target observation opportunities that interfere with area mapping opportunities and some that do not, we want to prefer those that do not. We can accomplish this by using a two-pass algorithm. In the first pass, we identify all the snapshot area map opportunities and schedule only target observations that don't interfere with those times. If we manage to schedule two validation target observations per day in the first pass, we're done. Otherwise, in the second pass, we continue to schedule validation target observations regardless of whether they interfere with other opportunities up to two per day. In this way we can ensure we schedule exactly two validation targets per day whenever possible. Once these two passes of scheduling target mode observations are complete, the scheduler continues adding lower priority observations to the schedule consisting of snapshot area map, nadir, and glint modes.

Determining Visibility

OCO-3's field of view is limited due to obstructions by other components of the ISS. The scheduling software must account for the limitation without sacrificing too much run time or precision. The occlusion mask is defined as a set of polynomials for several longitude segments. Checking if a point is contained within the mask is constant time.

We currently check visibility for area map mode and target mode. For target mode, the region is the line that will be repeatedly scanned over. For area map mode, the region is the 80 km square surrounding the center point of interest. As a result of not modeling the PMA movement, there may be cases where we can't be certain whether or not a target will be visible at a certain time. Our answers will therefore

be one of three possibilities: "certainly visible," "certainly not visible," and "possibly visible." Our policy is to schedule only observations of targets that are certainly visible.

Problem

Given inputs:

- Target: a set of points on the Earth's surface
- Time range: start and end times of the desired observation window
- Visibility set (V): A set of azimuth/elevation points in the satellite-centered reference frame that describe the instrument's visibility.

Determine whether the target is visible to the instrument for the entire time range.

Solution

The visibility mask for OCO-3 is defined by a black-box function that maps an azimuth and an elevation to a boolean signifying whether or not that point is visible. We can visualize this mask by plotting the boundary points separating visible and non-visible points on a unit sphere.

This unit sphere plays a key role in determining whether or not a target is visible. Our approach is to project the target onto the unit sphere, convert the coordinates to azimuth and elevation, and then check for visibility using V. This process is repeated at multiple points in time to check that whether the target is visible for the entire time range. We define several variations on the approach that vary in how we project the target on the unit sphere and check that it's visible and in how we sample in time.

Checking Visibility To check if a target is visible at a particular point in time, we start by projecting the target onto the unit sphere around the satellite. This is done by computing the vector from the satellite to the target, translating it into a reference frame that has its origin at the satellite's position, and normalizing the vector. This gives a point on the unit sphere that can be checked for inclusion in V. For glint mode, this approach is satisfactory. But for snapshot area map and target mode, we do not know exactly where the PMA is pointed at any time. We consider three approaches to addressing this complication:

- Centroid: Project only the centroid of the target and check that the single point is in V.
- Corners: Project the corners of the target and check that all are in V.
- Configuration space: Define a configuration space that characterizes all points on the unit sphere that represent the centroid of a visible target. Project only the centroid of the target on the unit sphere and check that the point falls within the configuration space.

For area mapping mode, the corners are the four corners of square that defines the region, rotated to be parallel to the heading of the ISS. For target mode, the corners are the two points that define the line we will be sweeping over, which are also rotated to be parallel to the heading. The visibility checking functions take in either the $centroid$ or a list of $corners$ the start (st) and end (et) of the time range to check over, and a method for updating the time to check ($timeMethod$), which is discussed in the next section.

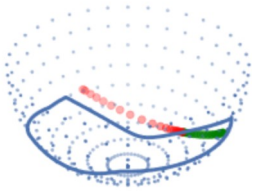

Figure 2: Tracing the visibility of a target's centroid over time.

The centroid approach has the obvious drawback that it can never say with certainty that an area map or target mode target is visible. At any time, even if the centroid is visible, it may be the case that one of the corners is not visible. But it has the additional drawback that it often can not say a target is not visible, either. If the centroid is not visible, then clearly it can not be certain the target is visible, but additionally it may be the case that the PMA is pointed at a part of the target other than the centroid, so neither can it be certain the target is not visible. All we can say with any confidence is that if the centroid is not visible for the entire duration of the observation, the target is almost certainly not visible, because at some time during the observation the PMA will be pointed at the target's centroid. Figure 2 shows an example of a centroid being tracked through time, with red showing times when the centroid is not visible, and green showing times when it is. Algorithm 1 shows an implementation of this visibility check.

```
Function Centroid.TargetIsVisible(centroid, st, et, timeMethod):
    t = st
    while t ≤ et do
        if not pointIsVisible(centroid, t) then
            return false
        end
        t = timeMethod.updateTime(t, et, centroid)
    end
    return true
```

Algorithm 1: Checking Visibility Using Centroid Method

The corners approach gives much more confidence in the results. For area map mode targets, these are the four corners of the bounding box defining the area, and for target mode targets, the corners are the top and bottom points of the line that will be traced out. If all corners are visible for the entire time range, we say the target is certainly visible. Depending on the shape of the visibility set, it should give zero or very nearly zero false positives. It also allows "certainly not visi-

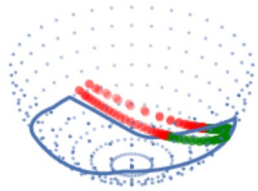

Figure 3: Tracing the visibility of a target's corners over time.

ble" answers if, at any time, none of the corners are visible. Figure 3 shows the visibility of the four corners of an area map mode target. Algorithm 2 shows an implementation of this visibility check.

```
Function Corners.TargetIsVisible(corners, st, et, timeMethod):
    for corner in corners do
        t = st
        while t ≤ et do
            if not pointIsVisible(corner, t) then
                return false
            end
            t = timeMethod.updateTime(t, et, corner)
        end
    end
    return true
```

Algorithm 2: Checking Visibility Using Corners Method

Our implementation of the configuration space approach will return either "certainly visible" or "possibly visible". We use a conservative definition of the configuration space. We first estimate the maximum angle between the projection of a target's centroid on the unit sphere and the projection of any other point within the target on the unit sphere as a function of the centroid's position on the unit sphere. Call this function $size$. We model $size$ as a trigonometric function with parameters that are found by performing least squares optimization. Now, the configuration space $P \subset V$ is the set of points $p \in P$ where the angle between p and the boundary point of the visibility set nearest to p is less than $size(p)$. Using this definition of a configuration space, there can be no false positives. There may be many false negatives, though, because we use a conservative (over-)estimation of the size of a target, and we don't faithfully represent the polygonal shape of the target. Algorithm 3 shows an implementation of this visibility check.

The centroid approach requires the least computation of the three. Only a single point must be projected onto the unit sphere, and only that point must be checked for inclusion in V. The four corners approach requires two or four times as many computations, depending on the type of target. All the corners must be projected onto the sphere, and all these

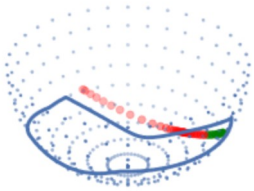

Figure 4: Tracing the visibility of a target's centroid within the configuration space over time.

```
Function ConfigSpace.TargetIsVisible(centroid, st, et, timeMethod):
    t = st
    while t ≤ et do
        if not pointInConfigurationSpace(centroid, t) then
            return false
        end
        t = timeMethod.updateTime(t, et, centroid)
    end
    return true
```

Algorithm 3: Checking Visibility Using Configuration Space Method

points must be checked for inclusion in V. The configuration space approach requires some amount of computation to be done up front to compute the configuration space, but then requires only a single point be checked for inclusion in the space.

Sampling over time To determine whether a target is visible over a range of time, we sample various points in time over that range. Increasing the number of samples may increase the confidence in our answer, but it will also increase the runtime of the algorithm. We consider two points in this space:

- Constant step: Take evenly spaced samples, so that after every sample a constant amount of time passes before taking the next.
- Adaptive step: Take larger steps when the target is near the middle of the visibility set and smaller steps when it's close to the boundary.
- Max step: Take as large a step as possible, so that only the start and end times of the observation are considered.

The constant step approach is the naïve approach. If the desired precision of the visibility window is one second, then the constant step size must be set to one second. Algorithm 4 shows an implementation of constant step.

We can do better by exploiting what we know about the visibility set and the motion of the target's projection on the unit sphere. Given any point projected on the unit sphere, we can compute the angle between that point and the nearest

```
Function ConstantStep.updateTime(t, et, point):
    return t + MIN_STEP
```

Algorithm 4: Updating time to check using Constant Step Method

point on the visibility set boundary. This angle is the minimum distance the projected point must travel before finding a visibility window boundary. We can empirically estimate the maximum angular velocity of the point and then divide the distance by this velocity to obtain a lower bound for the amount of time it will take to reach a visibility window boundary. To compute the adaptive step size, we find this time and then divide by a constant factor to account for uncertainty in the calculation. Algorithm 5 shows an implementation of adaptive step.

```
Function AdaptiveStep.updateTime(t, et, point):
    leeway = angleToMaskBoundary(point)
    step =
      max(MIN_STEP, leeway/ANGLE_VELOCITY)

    return min(et, t + step)
```

Algorithm 5: Updating time to check using Adaptive Step Method

The method for computing the adaptive step size is not ideal. The most glaring shortcoming is that the velocity of the target point's projection on the unit sphere will vary with its location on the sphere. When the satellite is directly over the target and the projection is on the bottom of the sphere, the velocity will be at its greatest. But when the satellite gets farther from the target, and the projection moves away from the bottom of the sphere, the velocity decreases.

We address this issue by finding the point on the visibility set boundary nearest to the target's projection, projecting that boundary point onto the Earth's surface, and then computing the distance between it and the target point. Given that we can compute an upper bound for the velocity of the satellite's nadir point, we can also compute the minimum amount of time that must pass before the target can possibly enter or leave the visibility set. We call this method the smart adaptive step approach. Algorithm 5 shows an implementation of smart adaptive step.

For completeness, the final approach we will consider is the max step approach. This approach trades rigor for speed. It requires the least computation, but it also gives us the least information about the target. If the target is not visible at either time, it's almost certainly not visible for the entire duration of the observation. If it is visible at both times, the target is possibly visible. Using this method, we can never say a target is certainly visible. Algorithm 7 shows an implementation of max step.

```
Function SmartAdaptiveStep.updateTime(t, et, point):
    nearestPoint =
      nearestPointOnMaskBoundary(point)
    pointEarth = pointOnEarth(point)
      nearestPointEarth =
      pointOnEarth(nearestPoint) distance =
      distanceBetween(pointEarth, nearestPointEarth)

    step = max(MIN_STEP,
      distance/DISTANCE_VELOCITY)
    return min(et, t + step)
```

Algorithm 6: Updating time to check using Adaptive Step Method

```
Function MaxStep.updateTime(t, et, point):
    return et
```

Algorithm 7: Updating time to check using Max Step Method

Results

Run time Figure 5 shows the average run time per area map mode targets for each of the approaches outlined above. There is not a considerable difference between the adaptive step and the smart adaptive step approaches because the smart adaptive step approach evaluates a target's visibility at fewer instances in time, but for each instance in time it requires slightly more work to determine whether a target is visible. The adaptive step approach's model of the motion of the target's projection on the unit sphere becomes less accurate the farther the projection is from the bottom of the sphere, so the longer the time range that we require the target be visible, the greater the benefit of using the smart adaptive step approach.

Effectiveness Figure 6 compares the effectiveness in determining the visibility of snapshot area map targets of the four corners, centroid, and configuration space approach, assuming the time step is computed using the constant step, adaptive step, or smart adaptive step approaches—all are equivalent.

Uncertainty in Orbital Ephemeris

While planning, we use an orbital ephemeris generated on a weekly basis. Due to the ISS being in Low Earth Orbit, there is considerable drag from the Earth's atmosphere. This results in the ISS drifting slightly from its predicted position. It is straightforward to account for this error in the computation of visibility windows. If we require that a target must be visible to an instrument for the time range $[st, et]$, and we know that our prediction for the time a location will be reached may be off by up to err, we can say that if the target is visible for the range $[st - err, et + err]$, it must be visible for the range $[st, et]$. Similarly, if the target is not visible over the range $[st + err, et - err]$, then it must not be visible over the range $[st, et]$.

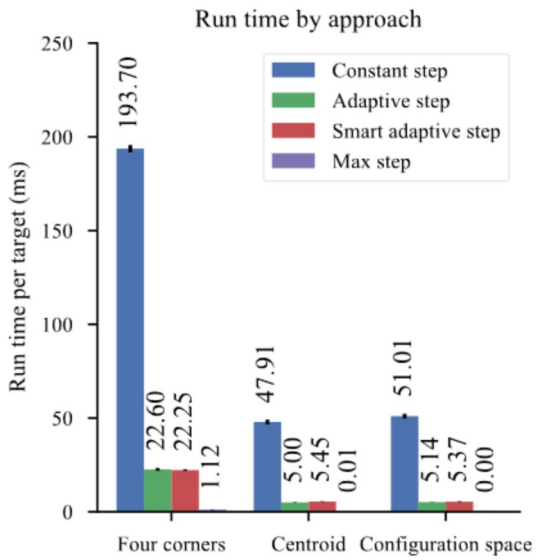

Figure 5: Average run time per target for each approach.

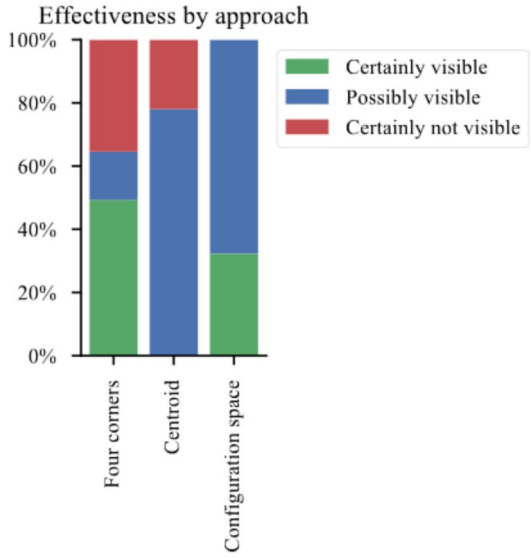

Figure 6: Effectiveness metrics for determining visibility of snapshot area map mode targets for each approach if not using max step.

We have found empirically that our prediction for the time when the ISS reaches a location may be off by up to three seconds. We require that every target must be visible for two minutes, so the majority of targets we could determine to be visible if there were no error in our prediction we also find to be visible accounting for error.

Final Approach

The approach being used operationally uses the four corners method with adaptive step, and accounts for three seconds of error in the ephemeris. Figure 7 compares the amount of snapshot area map targets that could be scheduled if there were no visibility constraints to the number of targets scheduled with the visibility constraints as well as accounting for three seconds of error.

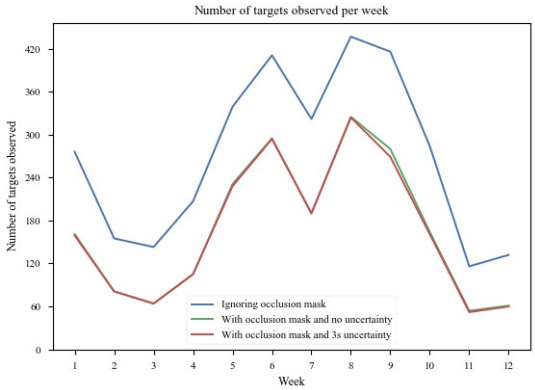

Figure 7: Graph comparing the number of snap shot area map targets that could be scheduled if there were no keep-out zones, the targets that could be scheduled with the keep-out zones and no error in the ephemeris, and the targets that could be schedueld with the keepout zones and accounting for error in the ephemeris.

PMA Calibration

Aside from its nominal science operation, the scheduling software for OCO-3 must also schedule the PMA calibration routine. Calibrating the PMA involves pointing the instrument at pre-defined points in the spacecraft's reference frame, taking images, and using the spacecraft's location to construct reference images against which the test images are compared. In order to ensure accuracy across the PMA's entire range of motion, the points are taken from a grid in the azimuth/elevation space where azimuth ranges from 0 to 340 degrees in 20 degree increments and elevation ranges from 30 to 80 degrees in 10 degree increments. To ensure we can construct quality reference images, we require the images be of land, at a point where the SZA is lower than some threshold.

There are two parts to scheduling PMA calibration: identifying the time windows during which each az/el point satisfies the above constraints in the first part, and scheduling

observations of these points while minimizing PMA movement (and time spent) in the second part. These problems are similar to those solved by CLASP, but CLASP does not currently directly support defining targets and constraints in different reference frames.

We start by creating a grid which serves as a discrete representation of the Earth's surface. Each grid point represents the terrain at the corresponding point on the surface - either land or water. We use this grid to identify the set of points on a land/water boundary, where a point is included in the set if any of the point's neighbors have a different terrain type than that of the point. We can then construct a distance field, where each point in the grid gives the distance to the nearest land/water boundary. This distance can be computed for each point by iterating over the set of land/water boundary points and calculating the distance for each using the haversine formula. It may be possible to do this more efficiently using R-trees (Schubert, Zimek, and Kriegel 2013), but we find this is not the bottleneck in the scheduler. We use the distance field to accelerate the search for time windows during which the instrument will be pointed at land when oriented toward a particular az/el point. Computing the point on the Earth the instrument is looking at is a relatively expensive operation, so we try to minimize the number of times we have to do it by sampling one point in time, determining the distance from the computed point to the nearest land/water boundary, estimating the time it will take to travel that distance, and then stepping forward that amount of time minus some safety margin.

Once we have computed the time windows subject to the land constraint, we can refine those windows to satisfy the SZA constraint. We sample the SZA at the target point at regular intervals over the range of each time window, restricting each window as necessary - or possibly splitting one into multiple windows - so that we select only time windows where the SZA at the target point is less than the constraint.

The second part of the scheduling problem is using the computed time windows to schedule calibration observations of each az/el point in the shortest time possible. The PMA takes time to switch targets, determined by the angular distance between the two targets, and our goal is to observe each az/el point while minimizing the time spent switching targets. To acquire high quality images, we require a minimum dwell time - the PMA must stay fixed on a particular az/el point for a minimum of this time before moving to the next target. We can simplify the problem by shortening the ends of our time windows by the dwell time - so that they represent the times during which we may start an observation - and by adding the dwell time to the time it takes to move between any two targets. We have thereby reduced the problem to an instance of the traveling salesman problem with multiple time windows, and we can solve the problem using standard heuristics for the traveling salesman problem (Hurkala 2015).

Future work

In the future we may have access to more precise predictions about the orientation of the satellite and the instrument. We can use this data to improve the accuracy of our snapshot area mapping target visibility analysis. Currently, we assume the visibility set is defined in a reference frame centered at the satellite and oriented such that one axis points directly toward the nadir point. In reality, the orientation may vary by up to a few degrees. We can account for this uncertainty by conservatively shrinking the visibility set, but this may rule out observations that would, in fact, be possible. We can improve our analysis by using the orientation data to adjust the reference frame in which the visibility set is defined.

In the adaptive step approaches to computing visibility windows, we come up with a conservative estimate for the earliest time the target's visibility status may change by only considering the distance between the target and the nearest point on the visibility set boundary. Instead, we could use our knowledge of the direction of the satellite's motion to determine approximately where on the visibility set boundary the target will next intersect. By doing so we could take more aggressive step sizes, reducing the number of samples required.

For the PMA Calibration, we currently only consider scheduling each point once. Once a schedule is found, there are still times before the end of the schedule when the visibility constraints are met, but no observations are scheduled. Adding in more observations after meeting the minimum requirements for the schedule can allow more samples for calibration.

Related Work

CLASP was previously used on the ground to schedule image acquisitions by the IPEX CubeSat (Chien et al. 2016). CLASP has been used for mission observation coverage studies for the upcoming Europa Clipper and JUICE missions (Troesch, Chien, and Ferguson 2017), and is being used to develop the observation plan for the NISAR mission (Doubleday and Knight 2014; Doubleday 2016). The ARIEL mission study (Roussel et al. 2017) also focused on long term observation planning. The ARIEL and Europa Clipper studies assume perfect knowledge of future ephemeris and certain execution of scheduled observations, which is appropriate for early mission design analysis, but not mission operations. This paper focuses on the mission operations use case, where there is some amount of uncertainty in the ephemeris.

CLASP was also used as a prototype for early stage mission planning of the THEMIS instrument on the Mars Odyssey spacecraft (Rabideau et al. 2010). The focus in the THEMIS study is performance of the squeaky wheel scheduling algorithm. This paper only considers a single pass of squeaky wheel when scheduling.

Conclusion

This paper has described the use of an automated scheduling system in development for the operations of the OCO-3 mission. We described each of the instrument's four operational modes and the scheduling of these modes. This involved adapting CLASP to achieve the desired results of only viewing two validation targets per day, and only viewing targets

within the designated safe zone. We outlined multiple approaches to the problem of determining target visibility and their respective results. We also described a scheduler for the instrument PMA calibration routine of imaging multiple azimuth and elevation points.

Acknowledgements

This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

Acton, C. 1996. Ancillary data services of nasa's navigation and ancillary information facility. In *Planetary and Space Science*, volume 44, 65–70.

Chien, S.; Doubleday, J.; Thompson, D. R.; Wagstaff, K.; Bellardo, J.; Francis, C.; Baumgarten, E.; Williams, A.; Yee, E.; Stanton, E.; and Piug-Suari, J. 2016. Onboard autonomy on the intelligent payload experiment (ipex) cubesat mission. *Journal of Aerospace Information Systems (JAIS)*.

Doubleday, J., and Knight, R. 2014. Science mission planning for nisar (formerly desdyni) with clasp. In *SpaceOps 2014*.

Doubleday, J. R. 2016. Three petabytes or bust: Planning science observations for nisar. In *SPIE 9881*.

Hurkala, J. 2015. Time-dependent traveling salesman problem with multiple time windows. In *Position Papers of the 2015 Federated Conference on Computer Science and Information Systems*, volume 6 of *Annals of Computer Science and Information Systems*, 71–78.

Knight, R., and Chien, S. 2006. Producing large observation campaigns using compressed problem representations. In *International Workshop on Planning and Scheduling for Space (IWPSS-2006)*.

NASA. 2019. Oco-3 https://ocov3.jpl.nasa.gov/ retrieved 2019-04-23.

of Technology, C. I. 2019. Tccon http://www.tccon.caltech.edu/ retrieved 2019-04-23.

Rabideau, G.; Chien, S.; Mclaren, D.; Knight, R.; Anwar, S.; Mehall, G.; and Christensen, P. 2010. A tool for scheduling themis observations. In *International Symposium on Space Artificial Intelligence, Robotics, and Automation for Space (ISAIRAS 2010)*.

Roussel, S.; Pralet, C.; Jaubert, J.; Queyrel, J.; and Duong, B. 2017. Planning the observation of exoplanets: the ariel mission. In *International Workshop on Planning and Scheduling for Space (IWPSS 2017)*.

Schubert, E.; Zimek, A.; and Kriegel, H.-P. 2013. Geodetic distance queries on r-trees for indexing geographic data. In *Advances in Spatial and Temporal Databases*, 146–164.

Troesch, M.; Chien, S.; and Ferguson, E. 2017. Using automated scheduling to assess coverage for europa clipper and jupiter icy moons explorer. In *International Workshop on Planning and Scheduling for Space (IWPSS 2017)*.