SpaceOps Conferences
28 May - 1 June 2018, 2018, Marseille, France
2018 SpaceOps Conference

10.2514/6.2018-2578

Spacecraft Block Scheduling for NASA's Deep Space Network

Timothy M. Hackett[*] and Sven G. Bilén[†]
The Pennsylvania State University, University Park, PA, 16802

Mark D. Johnston[‡]
Jet Propulsion Laboratory/California Institute of Technology, Pasadena, CA, 91109

Currently, NASA's Deep Space Network (DSN) is responsible for uplink to, downlink from, and/or tracking of dozens of missions for space agencies across the world. The DSN scheduling process starts about four months prior to the start of the schedule week, a process in which requirements are defined and then the schedule is created, de-conflicted, and negotiated over the next 2–3 weeks with a team of mission representatives. Now scheduled for late 2019, Exploration Mission 1 (EM-1) will deploy upwards of 12 SmallSat missions that will be served by the DSN. This will increase the DSN's actively serviced spacecraft by up to 30%, further increasing the difficulty of meeting all mission needs via the oversubscribed network. To mitigate their impact on DSN scheduling, a block scheduling process is proposed for scheduling the SmallSats. Block scheduling consists of aggregating spacecraft together into larger "pseudo-spacecraft" based on geometric alignment that then follow the same process as any other DSN mission to receive segments of track time. These tracks are then decomposed into tracks for individual users based on their specific requirements. This paper describes a full novel scheduling toolset for building candidate blocks, evaluating the efficacy of these blocks, and optimal and suboptimal de-blocking schemes. To demonstrate these developed tools, results from three simulations are presented: a blocking example with lunar SmallSats, blocking potential in the greater DSN spacecraft catalog, and opportunistic multiple spacecraft per aperture potential for the DSN spacecraft catalog. Block scheduling has the potential to reduce overhead and scheduling resources for the EM-1 SmallSats while also providing them with a better means to meet their mission requirements.

I. Nomenclature

B_i	=	total duration of blocking window i
$D_{u,r}$	=	nominal track duration of request r for user u
f_u	=	chosen instance of $g_x(i)$ for spacecraft u
$g_x(i)$	=	score of instance x of partial aggregate path i
R_u	=	the total number of tracks requested by spacecraft u
r	=	index of individual spacecraft request
S	=	the total number of spacecraft being analyzed in a simulation
U	=	the total number of spacecraft in a block
u	=	index of spacecraft in a block

II. Introduction

Currently, NASA's Deep Space Network (DSN) is responsible for uplink to, downlink from, and/or tracking of dozens of missions in low Earth orbit (LEO), high Earth orbit (HEO), and Deep Space for space agencies across the world. It consists of three complexes in Goldstone, CA, USA; Madrid, Spain; and Canberra, Australia, each containing one 70-m and three or four 34-m parabolic dish antennas [1]. Based on the geographical placement of these sites, at least one ground station is in view of a spacecraft at any given time [2]. The high demand for the DSN's services results

[*]Graduate Fellow, School of Electrical Engineering and Computer Science, 304 Electrical Engineering East
[†]Professor, School of Electrical Engineering and Computer Science, 313 Electrical Engineering East
[‡]Principal Scientist, Planning and Execution Systems Section, 4800 Oak Grove Dr.

in over-subscribed schedules, and users often have to accept less than their requested ground track times. The high cost and long build timelines for new DSN ground sites make it ever more important to minimize overhead and maximize site usage by optimizing each week's track schedule—both with algorithms and manual tweaking.

The DSN scheduling process [3] starts about four months prior to the start of the schedule week when the mission scheduling representatives enter their track requests into the Service Scheduling Software (S3) [4]. All user requests are compiled into one master schedule by the DSN Scheduling Engine (DSE), which tries to deconflict as many requests as possible given the flexibility (or lack thereof) in the various user requests. A "Builder of Proposal" (BOP), a human scheduler, then takes the output schedule and modifies it based on the context of each mission, as well as previous experience with the DSN schedule. The BOP generally eliminates hundreds of conflicts, but 10–20 conflicts generally remain afterwards. At this point, mission representatives use a peer-to-peer negotiation process in order resolve the final conflicts in the schedule, and then the schedule is baselined [5]. The process takes generally 2–3 weeks, so multiple schedule weeks are being negotiated at any given time [6].

In 2019, the Exploration Mission 1 (EM-1) is scheduled to deploy the first wave of SmallSat missions that will be serviced by the DSN. The details on which SmallSats being serviced are still being worked out, but upwards of twelve require S-band/X-band DSN services. An additional 12 spacecraft would increase the DSN's actively serviced spacecraft by up to 30%, which would make scheduling even more difficult on the oversubscribed network. Unlike the larger or "flagship" missions, such as Juno or Mars Science Laboratory (MSL), SmallSat teams do not have the resources to individually provide mission representatives to work alongside the current scheduling team. Furthermore, even if the SmallSat teams had the resources, adding such a large number of new mission users would be a major challenge to the current lengthy proposal/counter-proposal peer-to-peer negotiation process. As a result, a new approach to scheduling for the SmallSats needs to be developed in order to ensure they get adequate communications and tracking time [6].

In [6], the authors proposed two novel approaches for this issue: opportunistic gap scheduling and block scheduling. Opportunistic gap scheduling consists of inserting SmallSat track time into the gaps in the DSN schedule. Gaps in the schedule vary week by week depending on the spacecraft orbital trajectories and user requirements. Gaps tend to appear when spacecraft are all lined up in a certain portion of the sky (rather than relatively-evenly spread out). Filling in schedule gaps increases the network utilization without impacting existing mission schedules or scheduling processes, making it an ideal solution for SmallSats. For an example using three current lunar missions as a proxy for the upcoming SmallSats, there were very few gaps in the schedule that also aligned with the view periods of these SmallSats. Although a good method for supplementing track time, opportunistic gap scheduling was found to be insufficient to meet even modest scheduling demands.

Block scheduling consists of aggregating spacecraft together into larger "psuedo-spacecraft" that then follow the same process as typical DSN missions to receive blocks of track time. These blocks are then decomposed into schedules for the individual users based on their individual requirements. Unlike opportunistic gap scheduling, block scheduling will have an impact on existing missions but is effective regardless of the week-by-week relative spacecraft positions in the sky. However, block scheduling can help to lower overhead time attributed to setup and teardown—typically, 45 and 15 minutes, respectively. Overhead time includes both mechanical processes (e.g., slewing the antenna and powering up transmitters/receivers) and software processes (e.g., loading and configuring link operator monitoring software). A block as a whole would have one 45-minute setup time and 15-minute teardown time. When switching between spacecraft in a block, the setup time is expected to be reduced to about 15 minutes because the overhead reduces to mostly software processes. With many spacecraft in a block, significant overhead can be eliminated from the schedule. The block scheduling method provides better promise for meeting the spacecraft requirements [6].

A third method suggested in handling SmallSats and further optimizing the DSN schedule, in general, is using opportunistic multiple spacecraft per aperture (OMSPA) [7]. Multiple spacecraft per aperture (MSPA) is an existing service (used for the Mars missions) in which one antenna is used to simultaneously support downlink for multiple spacecraft those spacecraft that are within the same beam. Currently, four spacecraft can be downlinked simultaneously, whereas only one uplink is supported at a time [8]. MSPA significantly lowers the cost per aperture by servicing multiple spacecraft with the same aperture. The limitation with traditional MSPA, though, is the number of parallel receivers, which can be very costly. Unlike MSPA, OMSPA uses a single digital recorder per station to record all of the IF signals within the beam, which can then be demodulated/decoded in software at a later time. For missions in planetary orbits at Mars or beyond, all spacecraft are within one beamwidth for essentially the entire year; this does not hold true for lunar-orbiting spacecraft. OMSPA provides secondary spacecraft with more downlink opportunities if their spacecraft passes through the beam of the actively-tracked primary spacecraft. Being an asynchronous service, it does not provide uplink, real-time interactive, or two-way ranging support [7].

The work in [6] provided only a proof-of-concept for the blocking method. This paper considerably expands upon the blocking method by describing a full novel scheduling toolset for building candidate spacecraft blocks, evaluating the efficacy of these blocks, exporting these blocks to be scheduled into the master schedule, and then de-blocking the allocated block tracks back into the individual user tracks. Building upon the work in [6, 7], this paper leverages the developed toolset for exploring the blocking and OMSPA potential for the entire DSN catalog at large.

The following sections detail the development and analysis of the blocking algorithms. Section III provides general implementation details and the libraries used. Section IV provides the details of the blocking and deblocking algorithms. Section V provides simulations for blocking lunar SmallSats, blocking the entire DSN catalog, and OMSPA potential for the entire DSN catalog. Section VI provides future work and conclusions based on the work presented in this paper.

III. Libraries Used

This toolset was written in Python 3. Python's high-level development environment, large library support, and ability to handle large computational problems were the primary reasons for the selection. Pinover *et al.*'s work [6] was written in Javascript, which made it easy to develop a proof-of-concept simulation but clunky for scaling to large problem sets and datasets. The following subsections highlight the main libraries (not including the stock Python libraries) leveraged for this work.

A. SpiceyPy

Ephemeris files for DSN-supported spacecraft are available on the Service Preparation Subsystem (SPS) Webportal [9] as SPICE Spacecraft and Planetary (SPK) kernels. The SPICE (Spacecraft, Planet, Instrument, Camera-matrix, Events) Toolkit [10] is a powerful software package developed by NASA's Navigation and Ancillary Information Facility (NAIF) for analyzing geometric events and observations for any object with an ephemeris. NAIF provides the SPICE Toolkit in C, Fortran, IDL, and MATLAB, but does not provide currently Python support [11]. SpiceyPy [12] is an independent Python wrapper for the C SPICE Toolkit written by Andrew Annex. Being a wrapper, SpiceyPy provides the ability to code in high-level Python but provides the performance gains from low-level C [12]. In this work, SpiceyPy is used to calculate the ground station azimuth and elevation pointing angles towards the DSN-supported spacecraft.

B. Pandas, Selenium, and wget

To minimize the work required by the user, ephemeris files are automatically downloaded from the SPS Webportal. In order to do this, Pandas (Python Data Analysis Library) [13], Selenium [14], and wget [15] are used in conjunction. Selenium provides automated browser navigation, Pandas provides HTML parsing, and wget provides a mechanism to download the ephemeris files.

C. SciPy

SciPy [16] includes NumPy [17]—the defacto standard for numerical arrays and matrices in Python—which was used for array structures, array operations, and mathematical operations. SciPy also includes Matplotlib [18], which provides a plotting API similar to MATLAB and was used for plotting schedules and intermediary results.

D. Dict to XML

Dict to XML [19] greatly simplifies the process of exporting data stored in a Python dictionary structure. This library was leveraged for writing blocks and simulation parameters to XML files. These XML files can then be opened with any XML reader for further parsing and filtering.

IV. Algorithms

The algorithms developed can be split into blocking and deblocking ones. The blocking algorithms acquire the ephemeris files; calculate ground station pointing positions and filter by the blocking angle; shape the pseudo-spacecraft requests to fit the valid blocking time windows and attempt to find a valid schedule; and then exports these blocks to XML files for post-processing. The deblocking algorithms take in schedule-assigned block tracks and assigns individual spacecraft to portions of these tracks by either maximizing the minimum user satisfaction or (sub-optimally) fitting them to the track lengths. The blocking and deblocking algorithms are independent scripts and have no dependence on each other, which makes them more applicable to other applications (such as OMSPA).

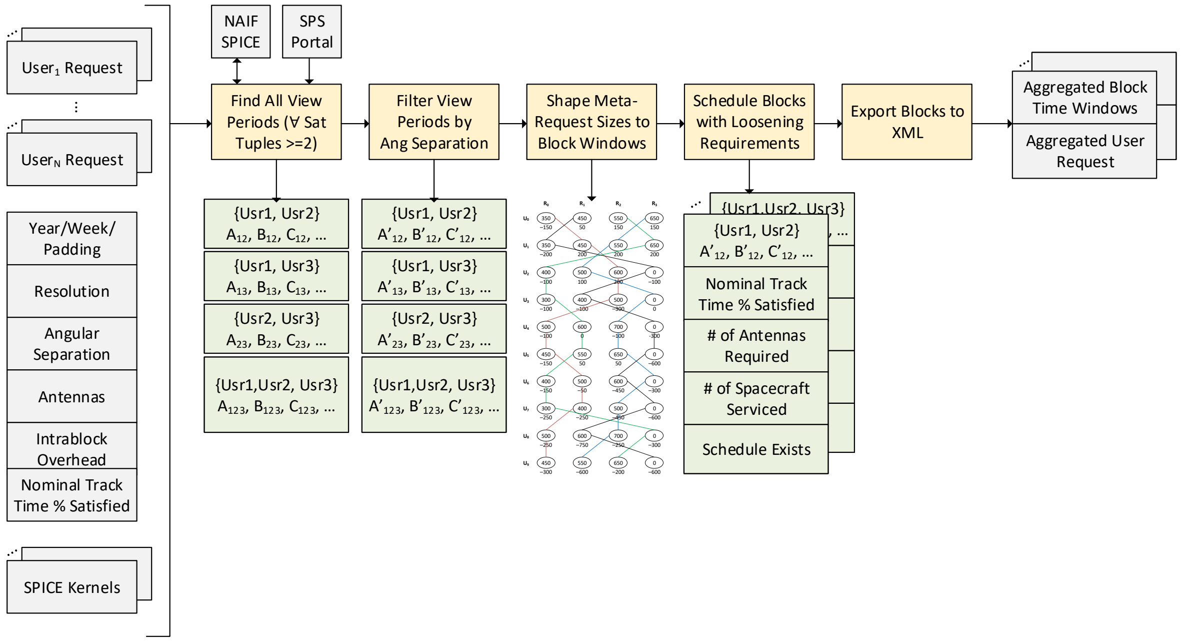
User₁ Request
Userₙ Request
Year/Week/Padding
Resolution
Angular Separation
Antennas
Intrablock Overhead
Nominal Track Time % Satisfied
SPICE Kernels
NAIF SPICE
SPS Portal
Find All View Periods (∀ Sat Tuples >=2)
Filter View Periods by Ang Separation
Shape Meta-Request Sizes to Block Windows
Schedule Blocks with Loosening Requirements
Export Blocks to XML
Aggregated Block Time Windows
Aggregated User Request
{Usr1, Usr2} A12, B12, C12, ...
{Usr1, Usr3} A13, B13, C13, ...
{Usr2, Usr3} A23, B23, C23, ...
{Usr1,Usr2, Usr3} A123, B123, C123, ...
{Usr1, Usr2} A'12, B'12, C'12, ...
{Usr1, Usr3} A'13, B'13, C'13, ...
{Usr2, Usr3} A'23, B'23, C'23, ...
{Usr1,Usr2, Usr3} A'123, B'123, C'123, ...
{Usr1,Usr2, Usr3}
Nominal Track Time % Satisfied
# of Antennas Required
# of Spacecraft Serviced
Schedule Exists

Fig. 1 General flow block diagram of the blocking algorithm.

A. Blocking Algorithm

Figure 1 shows a block diagram of the spacecraft blocking algorithm. The first portion of this diagram is the user inputs into the system, which can be classified into schedule timing, objects to be analyzed, and blocking parameters. The schedule timing inputs include the starting week number and year, the number of weeks over which to iterate, the schedule padding to include on each side of the week (e.g., an extra twelve hours on each side to account for tracks that may carry over between week borders), and the scheduling time resolution (e.g., 15-minute resolution). The objects to be analyzed include the spacecraft (both a list of the DSN abbreviation and the NAIF ID code) and the antennas to be used in the analysis. The blocking parameters include the interblock overhead (i.e., setup and teardown time for the entire track), intrablock time (i.e., overhead to switch between spacecraft within a block), a listing of fractions of the nominal requested duration to be analyzed (e.g., 100%, 90%, 80%, 70%), and a minimum track time to be included as a block track. The purpose of the minimum track time is to filter out very small track times where the overhead dominates the track.

The blocking algorithm consists of four main steps: extracting pointing positions using SPICE and finding combinations of spacecraft that are within the specified blocking angle, using the resulting blocking time windows to find a sub-optimal solution of the best ordering of the requests to make aggregated requests, using a backtrack scheduling algorithm to attempt to schedule the aggregated requests on every combination of specified antennas and at specified fractions of the requested nominal track duration, and exporting the results to XML log files. The following subsections provide more details on these functions.

1. Extracting Pointing Positions and Angular Filtering

With the spacecraft to be included in the blocking analysis provided by the user, the first step is determining the appropriate ephemeris files required and downloading them from SPS WebPortal. Using Selenium's automated browser control, a browser is opened to the SPS WebPortal main homepage. Here, the user manually authenticates with their appropriate user credentials. Once the user authenticates, they send a return to the Python script to notify that browser is ready to be taken control of again. At this point, the Python script navigates to each spacecraft's ephemeris page, downloads the HTML table of ephemeris listing, parses it (using Pandas) for the newest and most accurate ephemeris file(s) including the requested week(s), and then downloads the SPICE SPK file(s) (using wget).

With the ephemeris files downloaded, SPICE is called through the SpiceyPy wrapper to create position vectors in {azimuth, elevation, range} tuples relative to each specified ground station of each spacecraft (at the specified schedule time resolution). The function returns two dictionary data structures (with antenna numbers as the keys): a dictionary containing Boolean array entries that indicate whether each spacecraft is in view at each time segment and a dictionary containing the angular position of each spacecraft at each time segment relative to the specified antenna. The former data structure is initially populated with all true values and is used for masking spacecraft and time segments when doing any filtering. The latter data structure is used for calculating the angles between spacecraft.

The next step is to determine simple viewing periods (VPs) of each spacecraft by masking the VP array for only when the elevation angle for each antenna is above a specified minimum elevation angle for each antenna. There is also support for downloading the spacecraft VP files from SPS WebPortal if the user wants to account for lunar and planetary occultations in the analysis. Now, for each possible grouping combination of spacecraft (from two spacecraft to the total number of spacecraft analyzed), the centroid of their positions (relative to the ground antenna) is calculated for each time segment. If the angular positions relative to the centroid falls within half the magnitude of the maximum blocking angle, then that grouping of spacecraft can be potentially blocked for that time segment. Once every time segment in a week is evaluated, it is converted into a "block" structure, which contains the spacecraft IDs, the ground antenna, an array of every blocking time interval (greater than the minimum track time specified), and placeholders for block metrics, such as total track duration. If the block does not have any time intervals greater than the minimum track time specified, it is eliminated. Any block created at this point is considered potentially-blockable. It meets the angular separation and minimum track-time requirements, but may or may not meet the user request requirements.

2. *Greedy Algorithm for Request Shaping*

Even with a small set of spacecraft like the DSN (roughly 30 active spacecraft), testing every combination of spacecraft starts to explode combinatorically quickly. A stress test case would be finding blocking groups for all 34 active spacecraft on the DSN—this results in over one billion different groupings to test! Python's itertools library provides a way to iterate over all of these combinations without having to store the list of combinations in memory. This prevents combinatoric memory explosion and allows for scaling to large problem sets. Two methods leveraging simple set theory are used to drastically cut down the number of iterations tested. The first technique is that each set of groupings to test is made up of the unions of every combination of two potentially-blockable groupings of size $n-1$, where $3 < n < S$ and S is the total number of spacecraft being analyzed. The union of every combination of two spacecraft groupings can produce duplicates. If the duplicate grouping was already found to be potentially-blockable, it is skipped. Unfortunately, if the duplicate grouping was not found to be potentially-blockable, it must be retested again because only the listing of potentially-blockable groupings are stored in memory, not all groupings—otherwise, there would be a memory explosion. Second, for all groupings of size n larger than two spacecraft, a spacecraft grouping is only evaluated if every subset of $n-1$ was found to be potentially-blockable.

After potential-blocks have been created based on VP and angular separation, these spacecraft groupings need to be evaluated if they have blocking time windows that meet the conglomerate spacecraft duration and inter-request gap-time requirements. For the general case, when missions request time on the DSN, their requests do not have order dependence (as long as the gap-time requirements are met). This creates the problem of how to best combine requests together into pseudo-spacecraft aggregated requests. Ideally, the requests should fit relative to the durations of the blocking windows—depending on the orbits of the spacecraft in the grouping, the length of these blocking windows may change throughout the week. Assuming that one aggregated request can have at most one request from each individual spacecraft, the number of aggregated combination requests is $[\max(R_u)]!^U$, where R_u is the total number of tracks spacecraft u is requesting and U is the total number of spacecraft in the block. As the number of users in the group increases, the number of possible aggregated request combinations explodes exponentially. For relatively small groups, say three spacecraft, it is reasonable to try every request. But, for a larger group of future CubeSats, say ten spacecraft, each asking for up to 4 tracks per week, this results in 1,048,576 different combinations to test. This is infeasible when there are hundreds or thousands of potential spacecraft groupings to be tested.

To combat this combinatoric explosion, a sub-optimal greedy algorithm was developed that converts the problem to $U[\max(R_u)]$ combinations to try, which now scales linearly with the number of users. Figure 2 shows a diagram of the algorithm. Along the horizontal axis are the request numbers (e.g., $r = 0, 1, \ldots, 3$). On the vertical axis are the spacecraft users (e.g., $u = 0, 1, \ldots, 9$ and $U = 9$) where the users are sorted by the number of individual tracks requested in decreasing order. Inside each node on a single row is the nominal duration requested for each request for that spacecraft. If a spacecraft does not have $\max(R_u)$ requests, that request duration is 0. For example, for $u = 0$, the request durations are 400, 500, 600, and 0 units for requests $r = 0, 1, 2, 3$, respectively. Given the length of the four blocking windows that the requests are trying to fit (e.g., $B_0 = 5000$, $B_1 = 5000$, $B_2 = 5000$, $B_3 = 3000$ units), each user is allocated B_i/U units of the blocking window (e.g., $B_0/U = 500$, $B_1/U = 500$, $B_2/U = 500$, $B_3/U = 300$ units). There are $\max(R_u)$ aggregated requests that need to be formed from the individual user requests. For each user, an aggregated request is assigned up to one of each user's requests, and no two aggregated requests can be assigned to the same individual user request. The new aggregate score for path i for a particular instance of new aggregate paths including user u is calculated by

$$g_x(i) = f_{u-1}(i) - B_i/U + D_{u,r}\,, \tag{1}$$

where $f_{n-1}(i)$ is the old score for aggregate request i (prior to adding user u), B_i/U is the allocated duration units per user for aggregate request i, $D_{u,r}$ is the duration of request r of user u, and x is the index of the particular instance of new aggregate paths. For initialization, $f_{-1}(i) = 0\ \forall\ i$. After all new aggregate path scores have been computed, the instance of paths chosen to add user u is the instance x that has the smallest maximum aggregate path score, or

$$f_u = g_v\,, \quad \text{where} \quad v = \underset{x}{\operatorname{argmin}} \left\{ \max_i \ g_x(i) \right\}\,. \tag{2}$$

After iterating through all users, the $\max(R_u)$ resulting aggregate paths are the assignments of user requirements to aggregate requirements. This assignment is not optimal, but provides a "good enough" solution for request shaping.

Implicitly, the greedy algorithm assumes that the number of blocking windows available is exactly the same as the number of aggregate requests required. If there are more available blocking windows than aggregate requests, then our algorithm takes $\max(R_u)$ representative sample window durations. This is done by splitting the number of windows into $\max(R_u)$ groupings and then taking the median duration of each group.

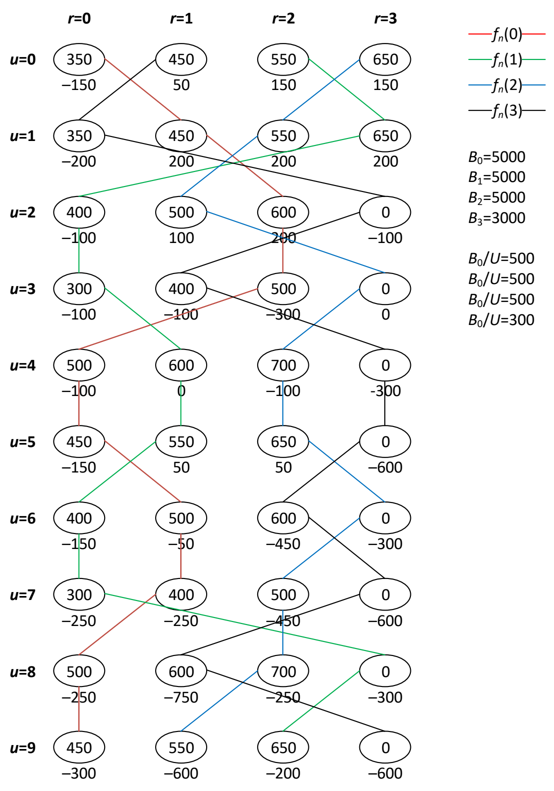

Fig. 2 Greedy algorithm example with 4 aggregate requests and 10 users.

3. Backtrack Scheduling

Once the spacecraft requests have been aggregated into pseudo-spacecraft requests, the next step is to determine if these requests can be satisfied by the blocking time windows. A backtrack scheduling algorithm is used to recursively attempt to find a valid schedule solution by incrementally building up partial solutions. The algorithm finds the first time window when the first aggregated request fits into the schedule. If a time window is found, then the algorithm adds the second aggregated request and finds the first location when this request fits in the schedule while meeting the gap requirements. If a valid time window is found for the second request, the algorithm adds in the third request, and so on. If a valid time window cannot be found for the second request, then the algorithm goes back to the first request and finds the next valid time window. It then tries to schedule the second request again. This algorithm continues until a valid schedule solution is found or the end of the schedule week is reached without a valid solution. If a valid schedule is found, the grouping of spacecraft is considered "blockable"; otherwise, the grouping is "not blockable".

Because the backtrack scheduling algorithm is scheduling the aggregated requests (and not the individual spacecraft requests), the maximum runtime of the algorithm is bounded by the maximum number of requests by a single spacecraft. As a result, this algorithm has the same maximum runtime bound regardless of the number of spacecraft in the block—assuming the same maximum number of individual requests from a single spacecraft. The average runtime may increase as more users are added because the probability of finding a valid schedule (and exiting the backtracking algorithm early) decreases as the durations of the aggregated requests increase.

The output of the backtracking algorithm is simply a Boolean value: a schedule does or does not exist for the block. Where the requests were scheduled into the blocking time windows is not important. The purpose of the algorithm is to identify spacecraft groupings that can be blocked together. The scheduling algorithm in the Service Scheduling

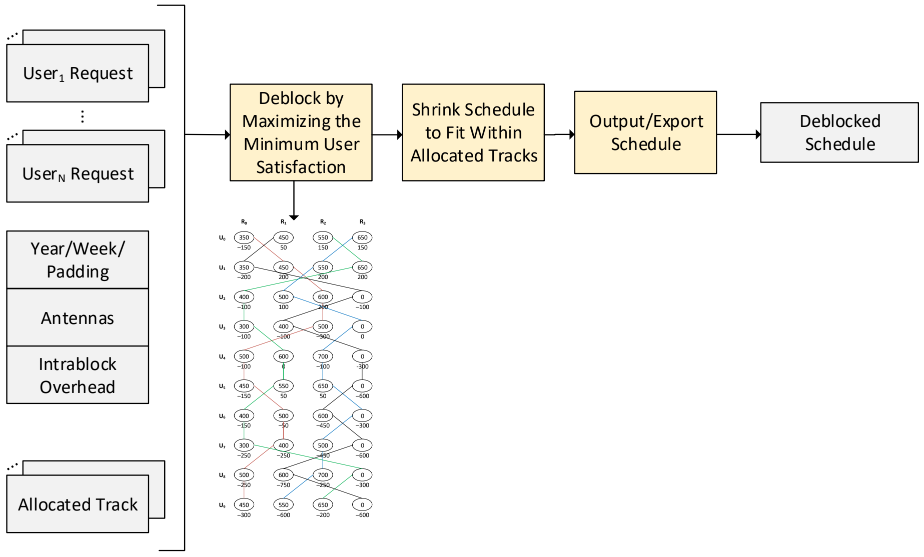

Fig. 3 General flow block diagram of the deblocking algorithm.

Subsystem (SSS) is responsible for scheduling this block into the master DSN schedule, as it takes into account all of other spacecraft in the DSN requesting tracking time.

Because the output of the backtracking algorithm is simply a schedule existence Boolean, it does not give any information on how the requirements could be loosened in order to make the spacecraft grouping be blockable or how the requirements could be tightened and still keep a spacecraft grouping to be blockable. To determine this information, the backtracking algorithm is run through a series of iterations with modified inputs. Specifically, the algorithm iterates through fractions of the aggregate request nominal track duration used (e.g., 100%, 90%, 80%, 70%) and every possible grouping of antennas specified (from a single antenna to all antennas). This produces a Boolean table for each potential block. This table can be useful to identify what resources are required for a potential block to be blockable at a certain level of user satisfaction. The BOP and mission scheduling team can also use this table to know the impact of changing a block's request duration when manually fixing conflicts in the full DSN schedule.

4. Exporting to XML

At the end of the blocking process, the attributes of blocks are exported to XML to be used with scheduling in SSS. These attributes include the blocking time windows (to be imported as the VPs in SSS) and the aggregate requests of the block. The goal of this software is to have a seamless integration with SSS—at the time of writing, all of the necessary interfaces are still being developed.

B. Deblocking Algorithm

There are two main ways of deblocking. For smaller groupings of spacecraft, an optimal brute force optimization scheme can be used in which the minimum user satisfaction is maximized. For any size groupings of spacecraft, the greedy algorithm described in Section IV.A.2 can be used to deblock by fitting (suboptimally) aggregate request sizes to the assigned track times. A block diagram of the deblocking process is shown in Figure 3.

1. Brute Force Maximization of Minimum User Satisfaction

For small groupings of spacecraft, a brute force optimization will find the globally optimal solution in a reasonable amount of time for any user satisfaction criteria. User satisfaction is defined as the ratio of total schedule time allocated to the spacecraft to the total schedule time requested. Through experimentation and analysis, maximizing the minimum user satisfaction was found to be the "most fair" goal. Using realistic problem sets, maximizing the mean user satisfaction often resulted in a large disparity between the most satisfied and least satisfied users. For example, three of four users

Table 1 Lunar SmallSat mission requirements derived from submitted ULPs.

	CuSP	BIOS	MLI
Number of Tracks	1	2	3
Earliest Start	0	0	0
Latest End	inf	inf	inf
Track Duration Min	0 hr	0 hr	0 hr
Track Duration Nominal	4 hr	2 hr	3 hr
Minimum Gap Time	1 day	1 day	1 day
Maximum Gap Time	5 days	5 days	5 days
Ephemeris	LRO	THB	THC

could have high satisfaction (e.g., 94%, 95%, 95.5%), whereas one user could be left with a low satisfaction (e.g., 30%). The mean satisfaction would be relatively high (e.g., 78.6%), which masks the fact that one user's requirements are not being satisfied well. Maximizing the minimum satisfaction implicitly creates a more equal prioritization.

The brute force algorithm traverses through every possible assignment of individual requests in the given track periods and calculates the minimum user satisfaction. As more users are added to a block, the number of potential schedule solutions explodes combinatorically. If runtime duration is not a major issue, this method could be used for medium group sizes of spacecraft.

2. Greedy Algorithm for Deblocking

If the brute force algorithm is unwieldy for the number of spacecraft in a block, the greedy algorithm described in Section IV.A.2 can be used. By attempting to minimize the maximum aggregated request score during each iteration (traversing through the list of spacecraft), the algorithm minimizes the worst-case shrinking that will occur to fit the aggregated request into allocated track time. Minimizing the maximum request shrinkage is equivalent to maximizing the minimum user satisfaction. This is because a user satisfaction less than 100% is due to the fact that the request duration is shrinking in order to fit into the allocated track time. Although suboptimal, the greedy algorithm is advantageous in scaling up linearly rather than exponentially as the number of users in a block increase.

3. Exporting to Text

The output of the core deblocking algorithms is the assignments of individual spacecraft requests to the allocated tracks. At the time of writing, the final step in deblocking is exporting the schedule (including setup, teardown, and intrablock overhead time) to a human-readable text file. When the deblocking software is integrated with SSS, it will modify the master DSN schedule in SSS to show the individual spacecraft in the time periods allocated for the block.

V. Simulations

To illustrate the versatility of the developed blocking and deblocking toolset, three simulations are presented in the following sections. The first simulation demonstrates the originally intended purpose of the software: blocking and deblocking SmallSats. The second and third simulations explore the potential for blocking and OMSPA, respectively, for the entire DSN spacecraft catalog over an entire year.

A. Sample Lunar Blocking Example

This simulation scenario explores the blocking potential for three of the SmallSats (Cubesat for Solar Particles (CuSP), BioSentinel (BIOS), Morehead Lunar Ice Cube (MLI)) to be deployed into orbit during EM-1. Table 1 shows the preliminarily-requested tracking requirements for the three spacecraft from a typical schedule week in their submitted user loading profiles (ULPs). Each spacecraft has different nominal track times and a different number of tracks. ULPs do not include gap times, so we make the assumption that the missions want their track times to be spaced out throughout the week. This is realized by a minimum and maximum gap times between tracks of one day and five days, respectively.

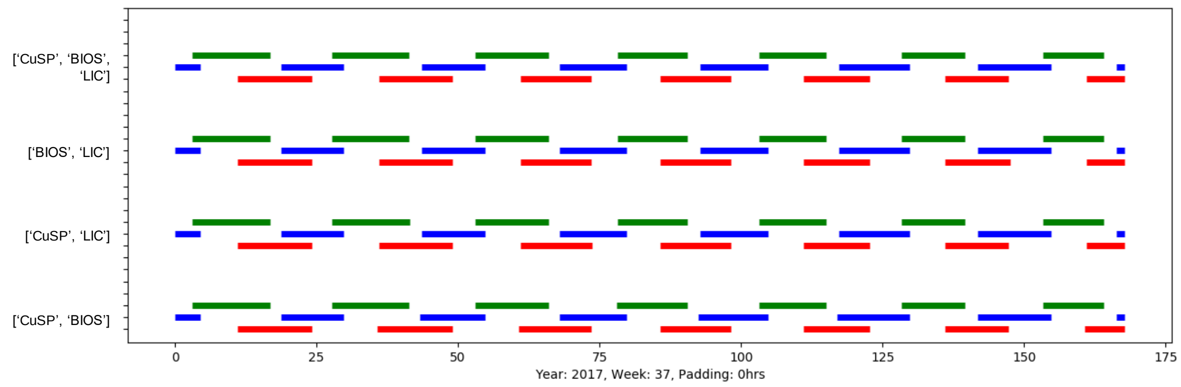

Fig. 4 Blocking time windows for all combinations of BIOS, CuSP, and MLI for Week 37 of 2017 for DSS-24 (red), DSS-34 (blue), and DSS-54 (green).

Because the expected launch date of EM-1 has slipped to December 2019, the full set of user request requirements (which would include gap time) is not currently available at the time of writing. For the same reason, the ephemerides for these SmallSats are not available yet. As a result, we use the ephemerides of Lunar Reconnaisance Orbiter (LRO), Themis B (THB), and Themis C (THC) from Week 37 of 2017 for CuSP, BioSentinel, and Lunar Ice Cube, respectively. LRO, THB, and THC are the same lunar proxy spacecraft used by Pinover *et al.* in [6]. We limit the SmallSats to be scheduled only on the 34-m Beam Waveguide 1 (BWG1) antenna at each DSN complex, which corresponds to DSS-24 (Goldstone), DSS-34 (Canberra), and DSS-54 (Madrid). During the blocking algorithm, we wish to explore blocking potential using 70%, 80%, 90%, and 100% of the requested nominal duration.

Running the blocking algorithms with these inputs, the potential blocking time windows for each combination of spacecraft grouping is shown in Fig. 4. The red, blue, and green bars represent the time in which the three spacecraft are within the blocking angle (±2.5°) of their pointing position centroid for DSS-24, DSS-34, and DSS-54, respectively. Between the three antennas, these spacecraft are always in view within the blocking angle. The four groupings of spacecraft listed are the four blocks that the blocking algorithm will investigate for blocking potential using the greedy and backtrack algorithms.

Table 2 shows the results of the backtrack algorithm. We can see that the grouping of all three spacecraft (along with every grouping of two spacecraft) is blockable using their originally requested nominal durations (and, thus, any fraction of this duration) and using any one antenna (and, thus, any grouping of antennas). Focusing on the block using all three antennas with 100% nominal track time, the aggregated nominal track requests are for 9.5, 5.25, and 3.0 hr, respectively (which includes 15-minute overhead time periods between spacecraft). This information along with the gap times and blocking time windows would then be imported into SSS and scheduled. The resulting track times would then be negotiated by the mission teams through the peer-to-peer service.

For the case of this example, let us assume that the three track times were negotiated to 427.5, 252.0, and 153.0 minutes from the originally requested 570.0, 315.0, and 180.0 minutes. These track times and the original user requirements are then fed into the deblocking algorithm using the greedy, suboptimal approach. Figure 5 shows the resulting deblocked schedule including setup, teardown, and intra-block overhead. The setup and teardown time is added outside of the blocking periods as the spacecraft do not need to be in view for these activities to occur. Using this deblocking method, the user satisfaction for THB, THC, and LRO were calculated to be 76%, 79%, and 73%, respectively. The low variance and high minimum satisfaction indicates a good solution.

B. Blocking Analysis on All Active DSN Spacecraft

Although the tool was originally intended for blocking SmallSat missions together, it can also be used to explore the blocking potential for the entire DSN spacecraft catalog. A simulation was set up to iterate from Week 33, 2017 through Week 33, 2018 with a ±2.5° blocking angle for 34 of the active spacecraft using the DSN with available ephemerides. These missions range from lunar missions to beyond the edge of the solar system. Antennas DSS-24, DSS-34, and

Table 2 Blocking analysis for BIOS, CuSP, MLI using DSS-24, DSS-34, and DSS-54 and analyzing 100%, 90%, 80%, and 70% of the requested nominal track duration.

Antenna	Nominal Duration %	Schedule Exists			
		{CuSP, BIOS, MLI}	{CuSP, BIOS}	{CuSP, MLI}	{BIOS, MLI}
{24}	100%	TRUE	TRUE	TRUE	TRUE
{24}	90%	TRUE	TRUE	TRUE	TRUE
{24}	80%	TRUE	TRUE	TRUE	TRUE
{24}	70%	TRUE	TRUE	TRUE	TRUE
{34}	100%	TRUE	TRUE	TRUE	TRUE
{34}	90%	TRUE	TRUE	TRUE	TRUE
{34}	80%	TRUE	TRUE	TRUE	TRUE
{34}	70%	TRUE	TRUE	TRUE	TRUE
{54}	100%	TRUE	TRUE	TRUE	TRUE
{54}	90%	TRUE	TRUE	TRUE	TRUE
{54}	80%	TRUE	TRUE	TRUE	TRUE
{54}	70%	TRUE	TRUE	TRUE	TRUE
{24, 34}	100%	TRUE	TRUE	TRUE	TRUE
{24, 34}	90%	TRUE	TRUE	TRUE	TRUE
{24, 34}	80%	TRUE	TRUE	TRUE	TRUE
{24, 34}	70%	TRUE	TRUE	TRUE	TRUE
{24, 54}	100%	TRUE	TRUE	TRUE	TRUE
{24, 54}	90%	TRUE	TRUE	TRUE	TRUE
{24, 54}	80%	TRUE	TRUE	TRUE	TRUE
{24, 54}	70%	TRUE	TRUE	TRUE	TRUE
{34, 54}	100%	TRUE	TRUE	TRUE	TRUE
{34, 54}	90%	TRUE	TRUE	TRUE	TRUE
{34, 54}	80%	TRUE	TRUE	TRUE	TRUE
{34, 54}	70%	TRUE	TRUE	TRUE	TRUE
{24, 34, 54}	100%	TRUE	TRUE	TRUE	TRUE
{24, 34, 54}	90%	TRUE	TRUE	TRUE	TRUE
{24, 34, 54}	80%	TRUE	TRUE	TRUE	TRUE
{24, 34, 54}	70%	TRUE	TRUE	TRUE	TRUE

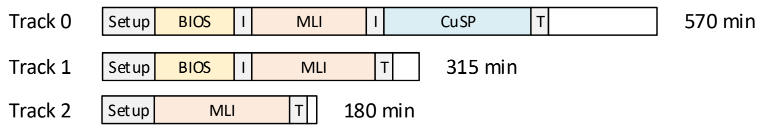

Fig. 5 Example deblocked schedules for BIOS, CuSP, and MLI. "Setup", "I", and "T" indicate setup time, intrablock overhead, and teardown time, respectively. The white bar indicates the extra track time requested but eliminated during the SSS scheduling and negotiation process.

DSS-54 were included in the analysis to provide a representative sample at each DSN complex. Any passes less than one hour were eliminated from a block. Because the blocking tool does not compute the link budget, it did not matter if a 34-m or 70-m antenna was chosen. To run through an entire schedule year (one week at a time), the simulation took approximately 7 hours on one logical core of an Intel Sandy Bridge i5 processor. Depending on the positions of the spacecraft in the sky, the time it takes to simulate one week is variable. When many spacecraft are located in the same part of sky, the simulation will take significantly longer because of a combinatoric explosion of possible groupings of spacecraft to test. With access to a large amount of processor cores, this simulation time could be reduced by running all of the weeks in parallel.

Table 3 shows a filtered selection of the spacecraft groupings that have the highest blocking potential—these spacecraft met the blocking angle requirement for a significant duration of time throughout the year. This analysis only explores the geometric alignment of the spacecraft; it does not take into account the individual spacecraft frequency band(s) or requirements. The resulting block database containing 7,858 blocks (an individual block is associated with a single antenna) was first filtered to only include passes in each block that are at least 8 hours in duration. This lowered the block count to 6,164. For a grouping of two spacecraft, this would give each spacecraft at least about 4 hours of track time. Spacecraft groupings whose superset is also a block and has the same number of passes were eliminated. This lowered the block count to 1,381 blocks. Of these 1,381 blocks, there were 495 unique spacecraft groupings (combining the antennas together).

The first column of Table 3 shows the spacecraft grouping. The second column shows the total duration of when these spacecraft are within the blocking angle across all three DSN complexes calculated only from passes that are at least 8 hours long. This total duration does not double count the time periods in which the spacecraft grouping is visible by two DSN complexes simultaneously. The third column shows the antenna for each spacecraft grouping with the largest number of passes (fourth column). The total duration metric is good for preliminarily identifying useful blocks, whereas the number of passes provides a more relevant number for the number of passes that could benefit from blocking.

At the top of the list are the expected groupings of spacecraft: the Mars, lunar, and Magnetospheric Multiscale (MMS) missions. The Mars missions already take advantage of their close proximity by using MSPA. The MMS missions are the only DSN spacecraft with the capability of being blocked today. As the LRO, THB, and THC missions were used as proxies for the SmallSat analysis, these were an obvious result of the analysis. Moving down the list reveals more interesting (and unexpected) potential blocking groupings, such as the Solar-Terrestrial Relations Observatory Ahead (STA) and the Spitzer Space Telescope (STF). These two spacecraft are within the blocking angle for over 4,100 hours in a year, which corresponds to 170 passes with at least 8-hour durations on DSS-24 alone. As shown in the table, there is a significant potential for blocking. Further analysis will be conducted on how many of these passes would meet user requirements. For a full listing of the DSN spacecraft and their abbreviations, see Ref. [20].

C. OMSPA Analysis on All Active DSN Spacecraft

While blocking can save track time by eliminating much of the overhead time for each spacecraft, OMPSA provides a much larger benefit as spacecraft can downlink simultaneously to a DSN site. Like the blocking analysis, an entire schedule year was simulated from Week 33, 2017 through Week 33, 2018. DSS-24, DSS-34, and DSS-54 antennas were used for coverage from all three complexes. Instead of the blocking angle, the half-power beamwidth of these antennas for S-band (±0.1315°), X-band (±0.0385°), and Ka-band (±0.0080°) were used [21]. The same 34 active DSN-supported spacecraft were included in the analysis. With the same setup as with the previous section, each simulation (S-band, X-band, and Ka-band) took approximately 3 hours to complete.

Table 3 Selected blocking time window analysis (using ±2.5°) for 34 active DSN-supported spacecraft from Week 29, 2017 through Week 29, 2018.

Spacecraft Grouping	Total Duration of Blocking Windows (hr)	Antenna with Most Passes	Number of Passes
{M01O, MVN, MEX, MOM, MRO, MSL}	9055.25	54	378
{LRO, THB, THC}	8999.00	54	358
{MMS1, MMS2}	8714.75	54	329
{MMS3, MMS4}	8692.00	54	328
{STA, STF}	4100.25	24	170
{MMS1, MMS2, MMS3, MMS4}	2649.00	24	98
{WIND, HYB2}	1473.50	34	62
{STA, PLC}	1096.25	54	45
{SOHO, STA}	1058.00	24	45
{STF, PLC}	1003.75	54	42
{STB, PLC}	981.25	24	40
{WIND, DSCO}	950.00	24	39
{WIND, PLC}	858.75	24	36
{WIND, STB}	753.25	24	32
{STF, HYB2}	725.00	24	30
{STA, STF, PLC}	692.00	24	28
{MER1, STB}	658.25	24	27
{SOHO, WIND}	575.50	24	25
{M01O, JNO}	516.00	24	21

Table 4 shows the potential OMPSA results for S-band with a minimum pass time requirement of 6 hours. The table is filtered to only show missions with S-band transceivers. The two Mars missions supporting S-band have the highest total duration. The pairs of MMS missions also have a significant amount of time that could support OMSPA. Unfortunately, all four MMS missions use the same carrier frequency, so they cannot support (O)MSPA. At X-band and Ka-band (not shown in the table), only the Mars missions fall within the half-power beamwidth for the minimum 6-hour passes. In other words, OMPSA will not greatly benefit the current DSN missions at S-band, X-band, or Ka-band (besides the Mars missions).

Table 4 OMSPA time window analysis (using ±0.1315°) for 34 active DSN-supported spacecraft from Week 29, 2017 through Week 29, 2018. This table is filtered to only show spacecraft that have S-band transceivers.

Spacecraft Grouping	Total Duration of Blocking Windows (hr)	Antenna with Most Passes	Number of Passes
{MEX, MOM}	9066.25	54	379
{MMS1, MMS2}	8746.00	54	342
{MMS3, MMS4}	8646.25	54	341
{MMS1, MMS2, MMS3, MMS4}	1757.25	54	71

VI. Conclusion

Block scheduling has the potential to reduce overhead and scheduling resources for the EM-1 SmallSats while also providing them with a better means to meet their mission requirements. The blocking algorithms described in this paper find groupings of spacecraft within the specified blocking angle, combine their individual requests to match the shape of the blocking time windows, attempt to schedule their aggregated requests across the specified antennas, and relax the nominal track duration to identify scheduling potential. Once the track times for the blocks have been negotiated through the standard DSN process, an optimal or suboptimal deblocking strategy can be used to break up the allocated block tracks into individual scheduled tracks by maximizing the minimum user satisfaction. An example lunar simulation with SmallSat requirements was presented demonstrating the feasibility of blocking. Additionally, the blocking algorithm was applied to the entire DSN spacecraft catalog in order to identify blocking and OMPSA potential for other missions. Although the current DSN spacecraft did not show high potential for OMPSA, blocking may be able to save a significant amount of overhead. With the algorithms developed and tested, future work includes integrating the Python application with the SSS software and running the algorithms with all of the SmallSat ephemerides (once available).

Acknowledgments

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. This work was also supported by a NASA Space Technology Research Fellowship (grant number NNX15AQ41H).

References

[1] "Deep Space Network," Online, 2018. Available at `http://deepspace.jpl.nasa.gov`.

[2] Imbriale, W. A., "Large Antennas of the Deep Space Network," *Deep-space Communications and Navigation Series*, edited by J. H. Yuen, Monograph 4, Jet Propulsion Laboratory, 2002, pp. 1–298.

[3] Johnston, M. D., Tran, D., Arroyo, B., Sorensen, S., Tay, P., Carruth, B., Coffman, A., and Wallace, M., "Automated Scheduling for NASA's Deep Space Networks," *AI Magazine*, Vol. 35, No. 4, 2014, pp. 7–25.

[4] Johnston, M. D., Tran, D., Arroyo, B., Sorensen, S., Tay, P., Carruth, J., Coffman, A., and Wallace, M., "Automating Mid- and Long-Range Scheduling for NASA's Deep Space Network," *SpaceOps 2012 Conference*, 2012, pp. 1–12.

[5] Carruth, J., Johnston, M. D., Coffman, A., Wallace, M., Arroyo, B., and Malhotra, S., "A Collaborative Scheduling Environment for NASA's Deep Space Network," *SpaceOps 2010 Conference*, 2010, pp. 1–12.

[6] Pinover, K., Johnston, M. D., and Lee, C., "Optimizing SmallSat Scheduling for NASA's Deep Space Network," *International Workshop on Planning and Scheduling for Space (IWPSS)*, 2017, pp. 124–132.

[7] Wyatt, E. J., Abraham, D., Johnston, M., Bowman, A., and Malphrus, B., "Emerging Techniques for Deep Space CubeSat Operations," *5th Interplanetary CubeSat Workshop*, 2016, pp. 1–17.

[8] Waldherr, S., "DSN Mission Support Definition and Commitments," *CubeSat Technical Interchange Meeting*, 2016.

[9] "Service Preparation Subsystem Portal," Online, 2018. Available at `https://spsweb.fltops.jpl.nasa.gov/`.

[10] Acton, C., "SPICE: An Observation Geometry System for Space Science Missions," Online, 2018. Available at `https://naif.jpl.nasa.gov/naif/index.html`.

[11] "An Overview of SPICE," Online, 2018. Available at `https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/pdf/individual_docs/03_spice_overview.pdf`.

[12] Annex, A., "SpiceyPy: The NASA JPL NAIF SPICE Toolkit Wrapper Written in Python," Online, 2018. Available at `https://github.com/AndrewAnnex/SpiceyPy`.

[13] McKinney, W., "Pandas: Python Data Analysis Library," Online, 2018. Available at `https://pandas.pydata.org/`.

[14] Muthukadan, B., "Selenium with Python," Online, 2018. Available at `http://selenium-python.readthedocs.io/`.

[15] Techtonik, A., "Wget: Pure Python Download Utility," Online, 2015. Available at `https://pypi.python.org/pypi/wget`.

[16] "SciPy: Scientific Computing Tools for Python," Online, 2018. Available at `https://www.scipy.org/`.

[17] "NumPy," Online, 2017. Available at `http://www.numpy.org/index.html`.

[18] "Matplotlib," Online, 2017. Available at `https://matplotlib.org/`.

[19] McGreal, R., "Dict To XML," Online, 2016. Available at `https://pypi.python.org/pypi/dicttoxml`.

[20] "DSN Current Mission Set," Online, 2017. Available at `https://deepspace.jpl.nasa.gov/files/DSNCurrentMissionSetMar82017.pdf`.

[21] Chang, C., *DSN Telecommunications Link Design Handbook*, Jet Propulsion Laboratory, 2015.