



Deliverable D1.3

Resource Aware Scheduling: Tool & Manual

Deliverable

<i>Number and title:</i>	D1.3 – Resource Aware Scheduling: Tool & Manual		
<i>Work package:</i>	WP1 (Energy and Resources)		
<i>Authors:</i>	Holger Hermanns (USAAR), Pedro D’Argenio (UNC)		
<i>Co-author:</i>	Juan Fraire (UNC)		
<i>Reviewers:</i>	Thomas Noll (RWTH), Leonardo Handsztok (INVAP)		
<i>Due date:</i>	M54	<i>Dissemination level:</i>	Public
	(31/03/2026)	<i>Version:</i>	1.0
<i>Submission date:</i>	01/05/2026	<i>Pages:</i>	19

Version history

<i>Version</i>	<i>Date</i>	<i>Notes</i>
1.0	01/05/2026	

Project

<i>Title:</i>	Models in Space Systems: Integration, Operation, and Networking		
<i>Acronym:</i>	MISSION	<i>Start date:</i>	01-10-2021
<i>GA no.:</i>	101008233	<i>Duration:</i>	63 months
<i>Call:</i>	H2020-MSCA-RISE-2020	<i>Website:</i>	mission-project.eu



This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233. This document reflects only the authors’ view. It does not represent the opinion of the European Union, and the European Union is not responsible for any use that may be made of the information it contains.

Contents

1	Executive Summary	4
2	Introduction	4
2.1	MISSION Context	4
2.2	Purpose and Scope	4
2.3	Relation to Earlier Work	5
2.4	Software Artifact	5
3	Resource-Aware Scheduling Problem	5
3.1	Inputs and Outputs	6
3.2	Objective and Constraints	6
4	Delivered Tool	7
4.1	Overview	7
4.2	Configuration	7
4.3	Inputs	8
4.4	Generic Mission Interface	8
5	Scheduling Model Implemented	8
5.1	Candidate Activities	9
5.2	Attitudes and Slewing	9
5.3	Battery and Solar Charging	9
5.4	Dynamic Programming	10
6	Reference Scenario: GomX-4A	10
6.1	Reference Run	11
6.2	Reading the Plot	12
7	User Manual	12
7.1	Requirements	12
7.2	Installation with uv	12
7.3	Installation with Standard Python Tooling	12
7.4	Running the Reference Scenario	13
7.5	Command-Line Overrides	13
7.6	Outputs	13
8	Examples	13
8.1	Toy Example	13
8.2	GomX-4A Example	14
9	Validation and Performance	15
9.1	Reproducibility	15

9.2 Feasibility Checks	15
9.3 Runtime and Memory	15
10 Limitations and Scope of Use	15
11 Publication and Reuse Plan	16
12 Using the Tool for Hand-Made Custom Missions	16
12.1 What the Tool Needs	16
12.2 Minimal Workflow	16
12.3 Small Improvements for Reuse	17
13 Conclusions	18
References	18
A Configuration Reference	18
B Command Reference	18
C Output File Reference	19
D Glossary	19

1 Executive Summary

This deliverable presents the MISSION resource-aware scheduling tool and its accompanying manual. The delivered software artifact is a standalone Python implementation of a generic dynamic-programming scheduler. It selects a subset of mission activities over a finite planning horizon while respecting access windows, attitude and slewing requirements, solar charging opportunities, and battery safety constraints. The repository includes GomX-4A as the reference scenario, but the scheduler itself is organized around generic input tables and configuration parameters.

The work is positioned in MISSION WP1, *Energy and Resources*. The project-level motivation is that New Space missions, and especially small satellites, must make operational decisions under tight resource budgets. MISSION addresses this setting through model-based tools for design and operation, and the tool documented here is a concrete scheduling artifact in that line of work. The document is self-contained: it first explains the resource-aware scheduling problem, then describes the model implemented in the tool, and finally provides a user manual for reproducing the reference run and adapting the tool to hand-made mission scenarios.

The reference scenario is the current GomX-4A scenario included in the repository. It uses an 18-hour scheduling horizon, from 10 May 2021 00:57:38 to 10 May 2021 18:57:38. In a local verification run performed for this draft, the dynamic-programming computation completed in approximately 208 seconds and produced the schedule plot shown later in this document. The resulting schedule selected all high-speed-link and ADS-B opportunities available after preprocessing, one of two NanoCam opportunities, and 26 of 61 inter-satellite-link chunks.

2 Introduction

2.1 MISSION Context

MISSION, *Models in Space Systems: Integration, Operation, and Networking*, develops model-based methods and tools for future space systems. The project is motivated by the operational and engineering challenges of New Space, where smaller and more cost-efficient spacecraft must still deliver reliable and effective services. The project website identifies the balance of available resources, in particular power consumption and data handling, as one of the central challenges for low-Earth-orbit missions and constellations.¹

This deliverable belongs to WP1, *Energy and Resources*. WP1 focuses on resource modeling, automated planning, optimization, evaluation frameworks, and knowledge transfer for the New Space context. Resource-aware scheduling is a direct instance of this work-package objective: it turns models of activities, energy, visibility, and operational constraints into an executable plan.

2.2 Purpose and Scope

The purpose of this document is twofold. First, it gives a compact explanation of the resource-aware scheduling problem for readers who are not expected to inspect the source code. Second, it documents the delivered tool as a usable software artifact: its scope, model, input and output files, the steps required to run it, and the interpretation of a representative GomX-4A run.

The deliverable is not intended to be a complete mission-operations handbook for GomX-4A. The repository used here is a standalone research and demonstration version of a generic scheduling tool, with GomSpace-specific interfaces removed. Its value for MISSION is that it captures a realistic resource-aware scheduling workflow in a public, reproducible, and inspectable form. GomX-4A is used because

¹<https://mission-project.eu/>

it is a credible and nontrivial reference instance, but the modeling interface is intentionally broader: a user supplies activity windows, solar exposure windows, activity values, attitude rules, and battery parameters, and the tool computes a feasible schedule with the maximum number of task scores without depleting the available resources.

2.3 Relation to Earlier Work

The algorithmic line behind this tool predates the MISSION project. A power-aware scheduling framework for LEO satellite fleets was presented in IEEE TCAD in 2020 [1]. That paper established the broader dynamic programming approach to managing fleets of LEO satellites under nonlinear resource constraints. A later Small Satellite Conference paper discussed the operational setting in which these techniques were integrated with GomSpace's Hands-Off Operations Platform and validated against GomX-4 operations [2].

This deliverable packages a simplified and standalone form of the scheduling tool for MISSION: the external operational interfaces have been removed, the scenario is distributed as CSV input data, and the document explains how the tool can be inspected, run, and adapted.

2.4 Software Artifact

The code documented in this deliverable is in the repository:

<https://dgit.cs.uni-saarland.de/Gregory/mission-resource-aware-scheduling>

At the time of drafting, the repository contains a Python command-line tool, GomX-4A input data, a default TOML configuration, plotting support, and a GitLab CI job that runs the tool and stores generated artifacts. The intended final publication path is to make this repository and the deliverable public.

3 Resource-Aware Scheduling Problem

Resource-aware scheduling is the problem of choosing a feasible and valuable set of activities from a larger pool of candidates. In a spacecraft setting, candidate activities may include payload operation, inter-satellite communication, ground-station downlinks, data collection, or housekeeping actions. Each activity is only available during certain time intervals and may require resources such as energy, pointing attitude, memory, or communication bandwidth. The scheduler must balance the value of executing activities against the need to remain within operational limits.

The tool documented here assumes that the time windows are already known. It does not decide where the spacecraft is, propagate an orbit, or compute a ground-station pass. Instead, it answers the next operational question: given a set of feasible-looking windows and a resource model, which activities should be selected to keep the complete plan feasible? This separation is what makes the tool generic. Orbit propagation, link-budget analysis, or payload opportunity generation can be performed by other mission-analysis tools, while this scheduler consumes their resulting windows in a simple tabular format.

For a small satellite, energy is often the most visible constraint. The spacecraft battery is depleted by background platform operations (on-board computer, thermal management, etc.), payload activities, and attitude maneuvers, and replenished by solar charging when the spacecraft is illuminated. Even if an activity has a high scientific or operational value, it may be infeasible if it would drive the battery below a safe operating threshold or require an attitude that conflicts with another selected activity.

3.1 Inputs and Outputs

The abstract input to the scheduling problem consists of:

- a scheduling horizon with start and end times,
- a set of candidate activities with start and stop times,
- a value or priority for each activity,
- resource-consumption parameters for activity execution and supporting actions,
- resource-production opportunities such as solar charging intervals,
- compatibility constraints such as attitude requirements, and
- safety constraints such as minimum and required-final battery state of charge.

The main output is a schedule, i.e., a decision indicating which candidate activities are selected. A useful scheduling tool should also produce diagnostics. For this deliverable, these diagnostics include per-activity CSV files with a scheduled/unscheduled flag, a battery state-of-charge trace, a battery load trace, a JSON export for downstream tools, a text summary of the solution, memory/runtime data for the dynamic-programming table, and a PDF plot.

The same abstract input/output view applies to GomX-4A and to a hand-made example. The names of the activities may change, the access windows may come from a different orbit, and the battery parameters may describe another spacecraft, but the scheduling contract is the same: activity windows and resource parameters go in; selected activities and resource traces come out.

3.2 Objective and Constraints

Let A be the set of candidate activities, let $x_i \in \{0, 1\}$ indicate whether activity $i \in A$ is selected, and let r_k denote the configured `cost_rate` for activity type k . Each activity has a value w_i , computed from the `cost_rate` of its activity type and its duration:

$$w_i = r_{\tau(i)} \cdot \Delta t_i,$$

where $\tau(i)$ is the activity type of i and Δt_i is its duration. The scheduler maximizes the total selected value

$$\max \sum_{i \in A} w_i x_i$$

subject to feasibility constraints over the planning horizon. In the delivered tool, these constraints are represented operationally rather than through a separate mathematical programming model. At each relevant time point, the dynamic program tracks the active tasks and the battery state, and rejects transitions that violate battery safety limits, the maximum load, or attitude compatibility.

This formulation deliberately keeps the objective simple. The delivered tool optimizes a weighted sum of scheduled activity durations, where the configured `cost_rate` supplies the weight per unit time for each activity type. Different missions may require richer objectives, such as explicit lateness penalties, fairness across payloads, data volume constraints, or robustness to uncertain access windows. Those extensions are outside the current artifact but fit naturally within the resource-aware scheduling perspective. The current objective is nevertheless useful because it allows each activity family to have a configurable priority via the `cost_rate` parameter. For example, high-priority downlink windows can be assigned a high `cost_rate`, while lower-priority payload or background-collection activities can be scheduled opportunistically when the battery trajectory allows.

4 Delivered Tool

4.1 Overview

The delivered tool is intentionally compact. The entry point is `main.py`. It reads configuration data, parses access-window tables, builds task objects, runs dynamic programming, writes output files, and generates a schedule plot. The tool is generic in its scheduling interface: the code does not require a specific spacecraft or payload name, but it does require the user to specify the mission as configured activity types and CSV windows. The implementation is organized as shown in Table 1.

Path	Purpose
<code>main.py</code>	Entry point coordinating configuration, scheduling, plotting, and export.
<code>runner/accesses/</code>	CSV parsing, window cropping, and conversion of selected activities back to output tables.
<code>runner/battery/</code>	Linear battery and kinetic battery model support.
<code>runner/config/</code>	Default TOML configuration and command-line override parsing.
<code>runner/dp/</code>	Dynamic-programming table, pruning, solution extraction, and trace generation.
<code>runner/plot/</code>	PDF plotting of access windows, battery state of charge, and load.
<code>runner/util/</code>	Date conversion, CSV utilities, and dataframe helpers.
<code>data/input/</code>	Reference GomX-4A scenario inputs.
<code>data/output/</code>	Generated outputs; ignored by Git except for a placeholder.

Table 1: Repository structure of the delivered scheduling tool.

The normal execution flow is:

1. parse defaults and command-line overrides;
2. read and crop solar exposure and experiment access windows;
3. generate candidate tasks with time, value, attitude, slewing, and power attributes;
4. construct the dynamic-programming time list;
5. fill and prune the dynamic-programming table;
6. reconstruct a best feasible schedule;
7. write text, CSV, and JSON outputs; and
8. generate the schedule PDF.

4.2 Configuration

The default configuration is stored in `runner/config/defaults.toml`. It defines the scheduling horizon, input and output paths, battery parameters, background and slewing power, experiment types, the solar exposure file, and attitude definitions. The configuration is deliberately readable: it is the main place where a user adapts the tool to a different scenario. In the current repository, this file is the GomX-4A reference configuration, but the same structure can describe another mission if the activity names, file names, priorities, attitudes, and battery parameters are changed consistently.

The current horizon is 18 hours. The battery configuration uses a total capacity of 299520000 mJ and an initial KiBaM state split between charge that is immediately available for discharge and charge that is stored in a bound reservoir. The configured safety floor is 0.55, and the required final state-of-charge fraction is 0.6. The configuration also defines the power drawn by each activity type, the `cost_rate` assigned to each activity type, and the solar charge associated with each attitude. These values are not

hard-coded into the algorithm. They are scenario data read at startup, which is why the same scheduler can be used for a hand-made mission instance once that instance is expressed in the expected TOML and CSV format.

4.3 Inputs

Input access files are CSV tables with start and stop timestamps in UTCG, i.e., Gregorian Coordinated Universal Time. For experiment windows, the parser ensures the windows are fully contained within the scheduling horizon after accounting for warm-up, cool-down, and attitude slewing. For solar exposure, the parser retains windows that overlap the horizon because partial-sun intervals still affect the battery trace.

Table 2 summarizes the input datasets included in the repository. The row counts exclude header rows and describe the raw files in `data/input`; the number of candidate activities considered by the scheduler may be smaller after horizon cropping and safety filtering.

Dataset	Rows	Meaning
HSL_SVAL	7	Svalbard high-speed-link access windows.
HSL_CDB	3	CDB high-speed-link access windows.
NanoCam_A	2	NanoCam payload windows.
ISL_A	83	Inter-satellite-link windows, represented as chunks.
ADSB_A	16	ADS-B payload windows.
Sun_A	473	Solar exposure windows.

Table 2: Input datasets included in the GomX-4A scenario.

4.4 Generic Mission Interface

For a reader who wants to adapt the tool, the most important point is that the scheduler sees a mission through three generic objects. First, it identifies activity windows: each row indicates that a given activity type can run between a start time and a stop time. Second, it considers resource and compatibility parameters: each activity type has a power demand, value rate, attitude requirement, warm-up or cool-down if needed, and an optional slewing cost via the attitude configuration. Third, it sees solar exposure windows and battery parameters, which determine whether a candidate sequence of activities remains safe over the full horizon.

The GomX-4A names in Table 2 are therefore examples of activity families, not conceptual requirements of the algorithm. A custom mission could use names such as CAMERA, DOWNLINK, THERMAL_CHECK, or ISL_PASS, provided that the corresponding CSV files and configuration entries are present. The scheduler does not need to know why a window exists; it only needs to know when the window exists and what resources are consumed or produced when it is selected.

5 Scheduling Model Implemented

The implemented model is a standalone adaptation of the resource-aware scheduling approach developed in the earlier LEO fleet-management work [1]. For this deliverable, the model is presented operationally, in terms of the data structures and files used by the public tool. This is the most useful view for users who want to run the software or create their own mission instances.

5.1 Candidate Activities

The parser converts access windows into task objects. Each task has:

- a start and stop time for the activity itself;
- a task type, corresponding to an experiment entry in the configuration;
- a value w_i , computed as the configured `cost_rate` for the task type multiplied by the activity duration;
- a task power demand;
- an attitude requirement; and
- a slewing interval before and after the activity when the relevant attitude demands it.

This conversion is the bridge from mission-specific input data to the generic scheduler. Once a row has become a task object, the dynamic program treats a GomX-4A downlink, a camera opportunity, or a hypothetical custom payload opportunity in the same way: as a time-bounded candidate with value, power demand, and compatibility constraints.

The configured activity types are shown in Table 3. The high-speed-link activities receive a much larger cost rate than the other task types in the current configuration. This makes the scheduler prefer downlink windows when the resource model allows them, while still using lower-value opportunities such as ADS-B and ISL when they fit.

Task type	Attitude	Power	Cost rate	Role
HSL_SVAL	HSL	6446 mW	100000000	High-speed link to Svalbard.
HSL_CDB	HSL	6446 mW	100000000	High-speed link to CDB.
NanoCam_A	NanoCam	403 mW	10	Camera acquisition window.
ISL_A	ISL	6430 mW	2	Inter-satellite-link chunks.
ADSB_A	Dummy	586 mW	1	ADS-B payload opportunity.

Table 3: Configured activity types in the GomX-4A reference scenario.

5.2 Attitudes and Slewing

The tool models the fact that not every activity can be executed in every spacecraft attitude. For example, high-speed-link windows and inter-satellite-link windows require specific attitudes, whereas ADS-B is configured to be attitude-independent. The configuration also includes a nominal attitude and per-attitude solar charge rates.

When an activity requires a nontrivial attitude, the task interval is extended by slewing time before and after the activity. During slewing, the tool accounts for additional power demand and compatibility restrictions. This is important because an activity can be infeasible not only during its active payload interval but also during the preparation and recovery intervals around it.

The current configuration marks the ISL_A activity type, shown in Table 3, as `slewless` (task-type identifier 3 in the configuration). This reflects that consecutive ISL chunks within the same attitude can be treated as a continuous operation without incurring repeated slew transitions between chunks. The dynamic-programming table, therefore, permits compatible consecutive chunks when their timing and attitude requirements line up.

5.3 Battery and Solar Charging

The battery model is central to the delivered scheduler. The implementation supports a simple linear model and the Kinetic Battery Model (KiBaM) introduced by Manwell and McGowan [3]. KiBaM represents the battery as two coupled charge reservoirs. The available reservoir contains charge that can

be used immediately by the spacecraft load, while the bound reservoir contains charge that is stored chemically and must transfer into the available reservoir before it can be discharged. This captures a rate effect: under heavy load, the available reservoir may become limiting even when some total charge remains; during lighter load or rest, charge can move back from the bound reservoir into the available one. This battery abstraction is one reason the scheduler is resource-aware rather than merely time-window-aware. Two activities may fit in the calendar but still be mutually infeasible if their combined demand drives the available charge below the configured safety threshold.

The tool computes the net load over each interval by combining background power, task power, slewing power, and attitude-dependent solar charging. If the spacecraft is in sunlight, the solar charge rate associated with the current attitude is added. If the spacecraft is in eclipse, no solar charging is applied. A transition is rejected if it exceeds the configured maximum load or drops either battery component below the safety threshold. At the final time, the selected schedule must also meet the required remaining state-of-charge threshold.

5.4 Dynamic Programming

The scheduler uses dynamic programming (DP) over a time list built from the scheduling horizon and all relevant task boundaries. It then builds a smaller DP time list from the start of task activation intervals. At each DP step, the algorithm considers compatible sets of active tasks, computes the battery change from the previous step, rejects infeasible transitions, and records the accumulated value. The DP algorithm is deterministic for a fixed set of inputs. This is useful for a deliverable and a public repository: the same commit and the same input files should produce the same schedule, modulo runtime differences across machines.

An entry in the DP table can be understood as:

$$(t, c, S) \mapsto v,$$

where t is a time-list index, c is the battery state, S is the set of active tasks, and v is the best accumulated value reaching that state. After processing a time point, the table is pruned. An entry is removed if another entry has at least as much charge, a subset of active tasks, and at least the same accumulated value. This dominance check is crucial for keeping the computation practical on the 18-hour GomX-4A scenario.

Once the table is filled, the tool scans the final time step for the best entry that meets the required final battery state. It then reconstructs the selected schedule by backtracking through predecessor entries. The selected task set is used to generate CSV outputs, battery traces, JSON data, and the schedule plot.

6 Reference Scenario: GomX-4A

The reference scenario is the current GomX-4A scenario included with the repository. It is the concrete instance used to demonstrate and validate the generic scheduler in this deliverable. It corresponds to the more complex scenario used for in-orbit experiments and is more demanding than the older GomX-4B scenario. The scheduling horizon is:

10 May 2021 00:57:38 to 10 May 2021 18:57:38.

The 18-hour horizon creates enough overlapping windows to exercise the scheduler and produce an interesting result, while remaining suitable for a reproducible demonstration. Throughout this section, names such as HSL_SVAL, NanoCam_A, ISL_A, and ADSB_A should be understood as configured task-family identifiers within a single scenario; in particular, ADSB_A denotes the ADS-B task family. The same scheduler could process a different set of names and windows without changing the core dynamic-programming algorithm.

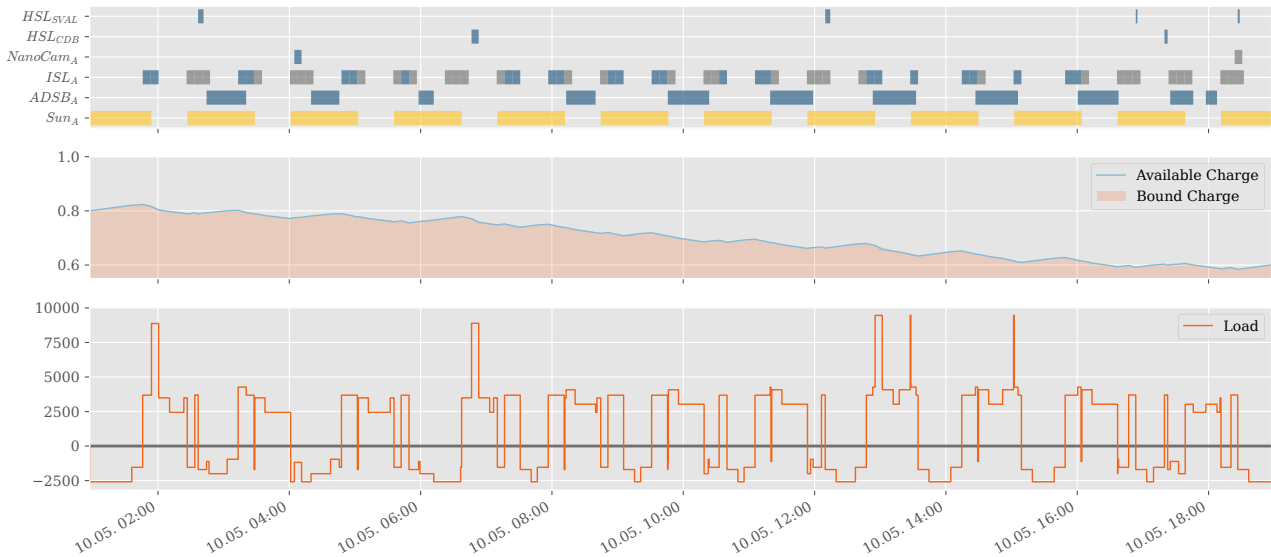


Figure 1: Output plot generated by the tool for the GomX-4A reference scenario. The top panel shows activity access windows and scheduled selections; the middle and lower panels show predicted battery state of charge and load.

6.1 Reference Run

A local verification run of the draft tool was performed with:

```
MPLCONFIGDIR=.matplotlib .venv/bin/python main.py
```

This run completed successfully after a defensive stack-limit change was applied for macOS compatibility. The dynamic-programming computation took 208.273 seconds in the local environment used for this draft. The result value was 171200046180, and the inverted value, i.e., the weighted value not realized by the selected schedule, was 34438.

The selected windows by task family are summarized in Table 4. The ISL opportunities are the main source of selection pressure: the tool schedules 26 of 61 candidate ISL chunks after preprocessing, while all available HSL and ADS-B opportunities are scheduled.

Task type	Scheduled	Interpretation
HSL_SVAL	4/4	All Svalbard high-speed-link opportunities selected.
HSL_CDB	2/2	All CDB high-speed-link opportunities selected.
NanoCam_A	1/2	One camera opportunity selected under resource constraints.
ISL_A	26/61	A subset of ISL chunks selected.
ADSB_A	11/11	All ADS-B opportunities selected.

Table 4: Preliminary selection counts from the verified local GomX-4A run.

The run also produced 132 state-of-charge trace points and 132 load-trace points. The dynamic-programming memory trace recorded a final cumulative entry count of 213094 and an approximate final cumulative byte count of 40709148. These values should be read as implementation diagnostics rather than mission metrics.

6.2 Reading the Plot

Figure 1 is the main human-readable output of the tool. The top panel lists the activity families and solar exposure. For activity families, selected windows (in blue) are plotted differently from skipped windows (in gray), allowing the user to see where the scheduler used or rejected available access opportunities. The solar exposure line indicates where charging opportunities are available.

The battery panel shows the predicted state of charge ahead. For the KiBaM model, the plot separates immediately available charge from bound charge that must first transfer into the available reservoir. The load panel shows the net load induced by scheduled activities, background consumption, slewing, and solar charging. Together, these traces explain why a schedule is feasible: the selected activity set is not just a list of windows, but a trajectory through the resource model.

7 User Manual

7.1 Requirements

The project targets Python 3.13 or newer. The declared runtime dependencies are:

- matplotlib,
- numpy,
- pandas,
- toml, and
- tqdm.

The repository includes `uv.lock`, and the author uses `uv` to manage dependencies. The code can also be run using standard Python virtual environment tooling.

On systems where Matplotlib cannot write to the default user cache directory, set `MPLCONFIGDIR` to a writable local directory before running the tool. This is not specific to the scheduler; it is a common Matplotlib configuration issue.

7.2 Installation with uv

The recommended installation path is:

```
git clone \
  https://dgit.cs.uni-saarland.de/Gregory/mission-resource-aware-scheduling.git
cd mission-resource-aware-scheduling
uv sync --locked
```

Using `uv sync -locked` ensures that the environment follows the dependency versions recorded in `uv.lock`. The reference CI job uses this path.

7.3 Installation with Standard Python Tooling

Users who do not use `uv` can run the tool with a virtual environment:

```
python3 -m venv .venv
. .venv/bin/activate
pip install -e .
```

For this draft, the repository metadata was adjusted so that editable installations discover `runner` and `main.py` without attempting to package the data directory as a Python package.

7.4 Running the Reference Scenario

With `uv`, run:

```
uv run main.py
```

With an activated virtual environment, run:

```
python main.py
```

If a local Matplotlib cache is needed, run:

```
MPLCONFIGDIR=.matplotlib python main.py
```

The program prints log messages and a progress bar while the dynamic-programming table is filled. At the end, it logs the runtime, result value, inverted value, selected task set, and plotting step. The full selected task list is written to `data/output/result.txt`.

7.5 Command-Line Overrides

The command-line interface currently supports the options listed in Table 5. These options are intended for quick experiments. For larger scenario changes, editing `runner/config/defaults.toml` is clearer.

Option	Type	Meaning
<code>-start</code>	string	Start time of the scheduling horizon.
<code>-end</code>	string	End time of the scheduling horizon.
<code>-init-soc-a</code>	integer	Initial available state of charge in mJ.
<code>-init-soc-b</code>	integer	Initial bound state of charge in mJ.
<code>-rem-soc</code>	float	Required remaining charge fraction at the end of the horizon.
<code>-tables</code>	string	Location of input access-window tables.
<code>-version</code>	flag	Print the command-line tool version.

Table 5: Command-line options exposed by the current tool.

7.6 Outputs

After a successful run, the output directory contains the files summarized in Table 6. The CSV and JSON files are intended for downstream inspection or integration; the PDF and text files are intended for quick human review.

8 Examples

8.1 Toy Example

A toy example helps separate the scheduling idea from the details of GomX-4A. Assume a small spacecraft has one battery, one sunlight interval, and three candidate activities, as in Table 7. Activities A and B overlap in time and require incompatible attitudes. Activity C is less valuable but occurs after the battery has partially recovered.

Output	Description
result.txt	Runtime, result value, inverted value, selected task set, and full solution list.
memory-usage.csv	Number of DP entries, approximate memory use, and elapsed time by DP step.
dp_trace.csv	Compressed load trace over the scheduling horizon.
battery_trace_soc.csv	Predicted battery state of charge.
battery_trace_load.csv	Predicted load trace.
Activity CSV files	One output file per activity type, with the Scheduled column filled.
schedule.json	Machine-readable schedule and battery/load prediction data.
schedule.pdf	Plot of activity windows, selected schedule, solar exposure, battery, and load.

Table 6: Generated output files.

Activity	Window	Value	Demand	Interpretation
A	00:10–00:20	10	high	Valuable activity during sun-light.
B	00:15–00:30	8	high	Conflicts with A by attitude and overlaps in time.
C	00:35–00:45	4	low	Fits after recovery and illustrates a later feasible task.

Table 7: Minimal toy example for resource-aware scheduling.

A scheduler that only considers value may try to select both A and B. A resource-aware scheduler rejects that combination because the activities overlap and require incompatible attitudes. It may select A and C if the battery remains safe after A and recovers sufficiently before C. It may instead select B and C if A’s timing or energy cost makes B the better feasible choice in the full resource trajectory. The important point is that feasibility is not local to a single activity: a schedule is judged as a path through time and resource states.

8.2 GomX-4A Example

The GomX-4A run is the realistic example delivered with the tool. It contains multiple task families with different priorities, power demands, and attitude requirements. The generated schedule demonstrates how the tool prioritizes high-value downlinks, selects lower-value opportunities when resources permit, and skips some ISL and NanoCam opportunities under the combined attitude and energy constraints.

For a general reader, the key result is not the exact selected task indices but the workflow:

1. define access windows and resource parameters;
2. run the scheduler;
3. inspect the scheduled/unscheduled decisions;
4. verify feasibility through battery and load traces; and
5. use the generated outputs for reporting or follow-up analysis.

9 Validation and Performance

9.1 Reproducibility

The repository includes a GitLab CI job that runs:

```
uv sync --locked
uv run main.py
```

and stores the generated log and output files as artifacts. This gives a straightforward reproducibility path: a user can compare the local output directory against the CI artifacts for the same commit.

For this draft, the tool was also run locally in a Python virtual environment. The run produced the expected artifact set: the schedule plot and JSON schedule, per-task CSV files, battery traces, a DP trace, and the text result summary. These outputs are sufficient to check that the tool is not merely computing a numeric objective but producing a concrete schedule with resource diagnostics.

9.2 Feasibility Checks

The implementation checks feasibility during DP transitions. A candidate transition is rejected if its battery update would cross the safety threshold, exceed the maximum load, or violate the active-task compatibility rules. After the final step, the scheduler only accepts solutions that meet the required remaining state-of-charge threshold.

For review, feasibility can be checked at three levels. First, each activity CSV indicates which windows were scheduled. Second, the battery and load traces show the resource trajectory induced by the selected schedule. Third, the schedule plot provides a quick visual check that the selected operations align with access and solar windows.

9.3 Runtime and Memory

The local reference run took approximately 208 seconds for the DP computation. The DP progress is not uniform: early steps are fast, while later steps can be significantly heavier because more combinations of active tasks and battery states are reachable. This behavior is expected for a dynamic-programming scheduler with pruning: the runtime depends on overlap patterns, the compatibility structure, battery-state diversity, and the extent to which pruning reduces dominated entries.

The generated `memory-usage.csv` file records the number of entries and approximate byte count at each dynamic-programming step. In the local run, the final cumulative entry count was 213094, and the final cumulative byte count was approximately 40709148. These figures provide a useful baseline for later scenario changes.

10 Limitations and Scope of Use

The current tool is a standalone research and demonstration artifact. It no longer contains GomSpace interfaces and should not be interpreted as a flight-operations product. It is intended to document and demonstrate a resource-aware scheduling workflow in a form that MISSION partners and external readers can run and inspect.

The current scope has several limitations. The input schema is fixed to the current CSV format. The access windows are deterministic. The objective is a weighted sum of scheduled activity durations. The model does not include onboard data storage, downlink volume, stochastic access uncertainty, or

an interactive user interface. The current code also contains scenario-specific details, such as a time adjustment for ADS-B output formatting. These limitations are acceptable for the present deliverable, but should be clearly stated when the repository is made public.

11 Publication and Reuse Plan

The deliverable and repository should be released together. The repository should contain a concise README with installation and quick-start instructions, while this deliverable provides the broader explanation, model documentation, and example interpretation. For final publication, the repository should include:

- a public URL,
- a license,
- a release tag or immutable commit hash,
- a passing CI job,
- the reference output artifacts, and
- clear instructions for reproducing the default GomX-4A run.

Draft finalisation note: Confirm the final repository visibility, license, and tag before freezing version 1.0 of the deliverable.

12 Using the Tool for Hand-Made Custom Missions

Although the repository ships with GomX-4A data, the tool can also be used as a lightweight scheduler for hand-made custom missions. The key requirement is that the custom mission must use the same generic interface as the reference scenario: CSV activity windows, a CSV sun-exposure table, and a consistent TOML configuration. This is useful for tutorials, sensitivity studies, internal reviews, and examples where the goal is to explain the scheduling logic without depending on a full orbit-analysis chain.

12.1 What the Tool Needs

A custom mission does not need to be called GomX-4A or use the same activity families. It does, however, need the pieces summarized in Table 8.

The activity-window CSV files use:

Access,Start Time (UTCG),Stop Time (UTCG),Duration (sec)

The sun-exposure CSV uses:

Start Time (UTCG),Stop Time (UTCG),Duration (sec)

All timestamps should follow the format already used in the reference data, for example:

25 Apr 2026 00:10:00.000

12.2 Minimal Workflow

The simplest custom workflow is:

Piece	How to provide it
Scheduling horizon	Set <code>schedule.start</code> and <code>schedule.end</code> , or override them with <code>-start</code> and <code>-end</code> .
Activity families	Add one <code>[[experiments]]</code> entry per activity type, with name, input filename, attitude, power demand, and value rate.
Activity windows	Create one CSV file per activity family using the expected access-window columns.
Solar exposure	Create one sun CSV file with sunlight intervals that overlap the scheduling horizon.
Attitudes and slewing	Define <code>[[attitudes]]</code> entries with slewing times and solar charge rates.
Battery model	Set capacity, initial charge, safety threshold, remaining-charge requirement, and KiBaM parameters.

Table 8: Minimal data needed to run a hand-made custom mission.

1. copy the existing `data/input/` directory to a new directory, such as `data/input-custom/`;
2. replace the CSV rows with hand-made windows for the desired activity families and sun intervals;
3. edit `runner/config/defaults.toml` so the activity names, file names, attitudes, power demands, and value rates describe the custom mission;
4. run the scheduler with the new table directory and horizon; and
5. inspect `schedule.pdf`, `result.txt`, `schedule.json`, and the generated CSV traces.

For example:

```
python main.py \
  --tables data/input-custom \
  --start "25 Apr 2026 00:00:00" \
  --end "25 Apr 2026 06:00:00"
```

This approach is sufficient for an illustrative mission in which the access windows are constructed manually. For instance, a six-hour low-Earth-orbit tutorial case could include periodic sunlight intervals, two ground-station downlink passes, three camera opportunities, and a few low-priority housekeeping windows. The schedule would not claim orbital fidelity unless those windows were generated by an orbit or access tool, but it would still be a valid demonstration of resource-aware scheduling.

12.3 Small Improvements for Reuse

Two minimal software improvements would make custom missions cleaner. First, the tool should accept a `-config` option pointing to a scenario-specific TOML file, such as `scenarios/custom-leo.toml`. This would avoid editing `runner/config/defaults.toml` when switching between GomX-4A and a custom mission. Second, the remaining scenario-specific output formatting, such as the ADS-B timestamp adjustment noted in the implementation, should be removed or guarded by configuration.

A further convenience, not required by the scheduler itself, would be a small helper script that writes synthetic CSV windows. Such a script could generate regular sunlight/eclipse intervals and user-defined activity windows, for example, in this deliverable or in future tutorials. This would keep the scheduler honest about its role: it consumes windows and optimizes resource-aware selections, while any orbit propagation or synthetic-window generation remains outside the core algorithm.

13 Conclusions

This deliverable documents a concrete resource-aware scheduling tool for MISSION WP1. The tool demonstrates how access windows, activity values, attitude constraints, solar charging, and battery safety constraints can be combined in a dynamic-programming scheduler. The GomX-4A reference scenario gives the deliverable a realistic basis: it is complex enough to exhibit meaningful scheduling trade-offs, yet compact enough to run as a reproducible software artifact. At the same time, the tool is not limited conceptually to GomX-4A; any mission that can be represented by the same access window and resource configuration interface can be used as an input instance.

The current draft should now be reviewed for technical accuracy, final repository metadata, and consistency with the planned public release. After review, the remaining work is to freeze the release commit, update the benchmark numbers from CI or the agreed reference machine, complete the cover dates, decide whether to add the `-config` convenience option, and remove the draft-finalization notes.

References

- [1] G. Stock, J. A. Fraire, T. Mömke, H. Hermanns, F. Babayev, and E. Cruz, “Managing fleets of LEO satellites: Nonlinear, optimal, efficient, scalable, usable, and robust,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3762–3773, 2020. [Online]. Available: <https://ieeexplore.ieee.org/document/9211417>
- [2] G. Stock, J. A. Fraire, H. Hermanns, E. Cruz, A. Isaacs, and Z. Imbrosh, “On the automation, optimization, and in-orbit validation of intelligent satellite constellation operations,” in *Proceedings of the AIAA/USU Conference on Small Satellites*, 2021, technical Session 5: Ground Systems, paper SSC21-V-05; author-generated version available as arXiv:2210.11171. [Online]. Available: <https://arxiv.org/abs/2210.11171>
- [3] J. F. Manwell and J. G. McGowan, “Lead acid battery storage model for hybrid energy systems,” *Solar Energy*, vol. 50, no. 5, pp. 399–405, 1993.

A Configuration Reference

The main configuration file is `runner/config/defaults.toml`. Its groups are:

- `schedule`: start/end time, weightless task types, and slewless task types;
- `paths`: input, output, and log directories;
- `battery`: capacity, initial state, safety thresholds, maximum load, and KiBaM parameters;
- `power`: background and slewing power;
- `experiments`: activity definitions, input files, attitude, warm-up/cool-down, power demand, and cost rate;
- `sun`: solar exposure input file; and
- `attitudes`: attitude names, slewing times, and solar charge rates.

B Command Reference

The command-line options are documented in Table 5. All date strings use the same format as the default configuration, e.g., 10 May 2021 00:57:38.

C Output File Reference

The generated output files are documented in Table 6. The most important review artifacts are the result summary, schedule plot, and JSON schedule. The CSV traces are useful for checking battery behavior and for producing additional plots.

D Glossary

Access window	Time interval during which an activity can be executed.
Activity	Candidate operation to be scheduled, such as a payload or communication window.
Attitude	Spacecraft pointing mode required by some activities.
Slewing	Transition into or out of an attitude, with time and power cost.
State of charge	Battery charge state used to determine energy feasibility.
KiBaM	Kinetic Battery Model, a two-reservoir battery abstraction with immediately available charge and bound charge that transfers into the available reservoir over time.
Scheduling horizon	Time interval over which the scheduler chooses activities.
Task value	Weight w_i assigned to an activity, computed as the configured <code>cost_rate</code> for the activity type multiplied by duration.
Inverted value	Weighted value of candidate activities not realized by the selected schedule.