TU Delft

Delft University of Technology

Document Version
Final published version

Licence
CC BY

Citation (APA)
Keller, M., Aleksandrov, D., Vanhamel, J., & De Smedt, V. (2026). Design of a Real-Time, Heuristic-Based Scheduling and Power Management Algorithm for a Re-Entry CubeSat. *Aerospace*, *13*(5), Article 445. https://doi.org/10.3390/aerospace13050445

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
In case the licence states “Dutch Copyright Act (Article 25fa)”, this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

*This work is downloaded from Delft University of Technology.*

aerospace

MDPI

*Article*

# Design of a Real-Time, Heuristic-Based Scheduling and Power Management Algorithm for a Re-Entry CubeSat

**Máté Keller [1,*], Daniel Aleksandrov [1,*], Jurgen Vanhamel [1] and Valentijn De Smedt [2]**

[1] Electronic Circuits and Systems Group, KU Leuven, 3000 Leuven, Belgium; jurgen.vanhamel@kuleuven.be
[2] Department of Electrical Engineering, KU Leuven, 3000 Leuven, Belgium; valentijn.desmedt@kuleuven.be
* Correspondence: mate.keller02@gmail.com (M.K.); dani.aleksandrov2001@gmail.com (D.A.)

**Abstract**

CubeSats are used as a platform in modern space missions due to their standardized form factor, reduced development cost, and shortened launch timelines. Earth observation, space weather monitoring and even re-entry applications make use of the CubeSat standard. Despite their advantages, CubeSats are constrained by limited onboard resources, with electrical power availability being one of the most critical bottlenecks. This work presents a dynamic, hybrid offline/online task scheduling and power management algorithm for a re-entry CubeSat, combining pre-computed schedules with real-time adaptation to changing flight conditions. The algorithm employs a heuristic-based approach, ranking tasks by parameters including priority, execution delay, duration, and power consumption. It adapts to varying flight conditions and system failures. In critical battery State of Charge (SoC) scenarios, only high-priority tasks above a defined threshold are executed, conserving power. A simulation suite was developed to evaluate performance under realistic mission profiles and stress tests with high loads and numerous tasks. The metrics included average and maximum task delay and average power consumption. The results show that appropriate heuristic weight selection can yield significant improvements in reliability and efficiency. The proposed algorithm offers a flexible, scalable solution for CubeSat power management, which is capable of maintaining operational reliability under dynamic conditions.

**Keywords:** CubeSat; scheduling; real-time; power management; re-entry; heuristic

## 1. Introduction

Since the dawn of the space age, satellites and unmanned probes have played a critical role in humanity's endeavor to explore the cosmos [1]. From communications satellites orbiting Earth, through remotely controlled space telescopes, to unmanned probes and rovers exploring other planets, uncrewed spacecraft have been at the forefront of human space exploration [2]. Historically, these crafts have mostly been multi-million-dollar pieces of equipment that very few organizations could afford to invest in [3].

Recently, however, CubeSats have been becoming increasingly popular due to their low cost and ability to play a large variety of different roles [4]. These small satellites can vary in size anywhere from a standard 1U CubeSat, which is a $10 \times 10 \times 10$ cm cube and weighs about 1 kg, to larger, up to 27U CubeSats, with a mass approaching 100 kg [5].

While unable to accommodate large experimental setups, CubeSats offer an affordable alternative to traditional scientific satellites, opening a vast array of possibilities for organi-

Check for updates

Academic Editor: Ausias Garrigos

Received: 28 February 2026
Revised: 21 April 2026
Accepted: 6 May 2026
Published: 9 May 2026

**Copyright:** © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

zations with lower budgets to perform experiments that would otherwise be too expensive to carry out.

Besides research applications, CubeSats also act as an affordable technology demonstrator platform. This is the basis for the Aether student team's CubeSat project, and the foundation for this paper [6]. Aether is a student team from the university of KU Leuven working on pioneering re-entry CubeSat technology by means of an inflatable heat shield.

A core component of any satellite is the on-board computer (OBC) and more specifically the flight software. The flight software, running on the OBC, is the brain of the satellite. It coordinates and manages the different tasks and systems on the spacecraft and thus represents one of the most critical components of any space mission. In this paper, a scheduling and power management solution is developed for the flight software, to work alongside its other components. The solution aims to be modular and easily customizable to be able to cater to a large variety of mission profiles.

This paper is structured as follows: in Section 2, a comparative analysis is done to evaluate the existing methods. In Section 3, the scheduling solution is introduced. In Section 4, the simulator and test scenarios are presented. Section 5 presents the results of this work. Finally, Section 6 concludes the paper with a discussion and potential for future work.

## 2. Choosing an Approach

This section reviews existing power-aware scheduling approaches for satellites, ranging from dynamic, real-time algorithms to offline, precomputed methods. The selected works are compared in terms of their trade-offs between schedule optimality, computational demand, and adaptability to changing flight conditions. The goal is to identify which elements from the discussed works best support the scheduling framework proposed in this paper and to motivate the choice of those elements.

Thakurta et al. [7] and Martinez et al. [8] present a heuristic-based approach in order to provide the system with the flexibility to accomplish the given tasks near optimally by adapting to the available parameters it is presented with, such as power input, orbital position, and battery state. Thakurta et al. [7] used conditional priority rules to dictate which operational mode should be active at a given time, with each mode assigned a fixed priority level based on its mission importance. This rule-based structure not only ensures that critical tasks are executed first, but also prevents the execution of two or more high-consumption modes, thus protecting the satellite from an undesirably high battery discharge rate. Martinez et al. [8] used a more complex heuristic, as this implements a more dynamic and mathematically structured method. By combining a task prioritization mechanism that adjusts priorities based on deadlines and execution history and a knapsack system to select the most beneficial set of tasks within the set power limits, this allows for an adaptive and flexible real-time algorithm that can perform near optimally while ensuring energy optimization.

Bernardo et al. [9] also used the knapsack system. The algorithm dynamically updates the task priorities in real time using a priority calculation that considers factors such as deadlines and completion percentages. Tasks with imminent or even missed deadlines are given overwhelmingly higher-priority values so that they are not left to hang and never be completed. The scheduling algorithm then searches the maximal total priority score of the selected tasks while also ensuring that the set power limits are maintained. The knapsack allows for improvements in meeting task deadlines and improving energy efficiency while maintaining a flexible real-time solution.

Rigo et al. [10,11] took a completely different approach compared to the discussed heuristics so far, known as the mathematical Integer Programming (IP). It has the objective

of maximizing task execution under the strict constraints of power availability, execution windows, activation limits, and minimum task durations. This approach produces globally optimal schedules, unlike the heuristics that the authors are aware of. Hence, it is more suited towards offline task scheduling plans, where the operational environment and the resources of the satellite can be predicted in advance. The downside is that the system is less adaptable overall to unexpected changes in flight conditions and requires recomputation of the entire schedule from the ground up to ensure optimality in the new flight conditions. In contrast, the heuristic-based strategies seen in [7–9], while not guaranteeing a globally optimal schedule, allow instead for a dynamic scheduling solution that adapts to the current flight conditions in real-time. This flexibility is crucial for our goal of creating a modular scheduling framework that can easily be tailored to different satellite configurations and mission profiles.

Another drawback of heuristic-based methods, presented by [7–9], are their heavy dependence on how their weights are defined. This means that poorly chosen and tuned weights can have undesirable consequences, such as inefficient task selection, missed deadlines, or over- or under-utilization of power resources, which are issues that IP solvers can often handle better. These models have strong mathematical guarantees, while the heuristic-based approaches miss this theoretical verification; instead, they must rely on simulation or Hardware-In-the-Loop (HIL) validation, as done in [9].

In conclusion, despite the variety of existing approaches, a clear trade-off remains. Mathematical optimization methods such as integer programming, as presented by Rigo et al. [10,11], provide optimal task schedules under predictable mission conditions, but lack real-time adaptability. In contrast, the heuristic-based methods outlined in [7,8], and [9] provide dynamic adaptability to real-time flight conditions but lack transparency in prioritization behavior and efficient tuning options for different mission profiles.

## 3. Custom Scheduler Design

The proposed design targets general space mission requirements and enables a modular approach. This creates a high level of adaptability to varying mission profiles while ensuring robust operation under dynamically changing power demands.

### *3.1. Overview*

A task, in the context of our algorithm, is a process that needs to be scheduled for execution at regular intervals. Table 1 describes the set of parameters every task must have.

**Table 1.** Task parameters.

| | Description | Unit |
|---|---|---|
| Name | A name given to the task for identification | String |
| Duration | The amount of time the task takes to execute | Seconds |
| Frequency | How often the task must be executed | Seconds |
| Power consumption | Amount of power consumed by the task when turned on | Watts |
| Interruptible | Whether a task can be interrupted if it has already started execution | Boolean |
| Priority | Relative priority of the task | 0–MAX_PRIORITY (=100 by default) |
| Active on re-entry | Whether the task should be executing during re-entry | Boolean |

These parameters ensure that every task is defined with great accuracy, and that all relevant information about the task is documented. The final parameter shown is an example of a mission-phase-specific boolean. Similar boolean parameters can be added whenever needed to define whether a task should be executed during other mission phases or operational profiles.

The initial set of tasks to be executed is uploaded to the OBC of the satellite on the ground. After launch, the task set can still be modified when the satellite communicates with the ground station. Tasks can be added or removed at the ground crew's discretion.

Once in orbit, the scheduler runs every M milliseconds. As a result, unexpected events, such as reduced power availability or subsystem anomalies, are incorporated into the next scheduling step. To support this process, the scheduler maintains a set of internal parameters for each task:

- Time_until_next_exec: Seconds remaining before task activation.
- Is_on: Binary flag indicating if the task is currently executing.
- On_timer: Duration (in seconds) the task has been active for.
- Is_in_ready_list: Binary flag for readiness status.
- Heuristic: The task's computed heuristic value.

Based on these parameters, every time the scheduler runs, it follows the flow shown in Figure 1.

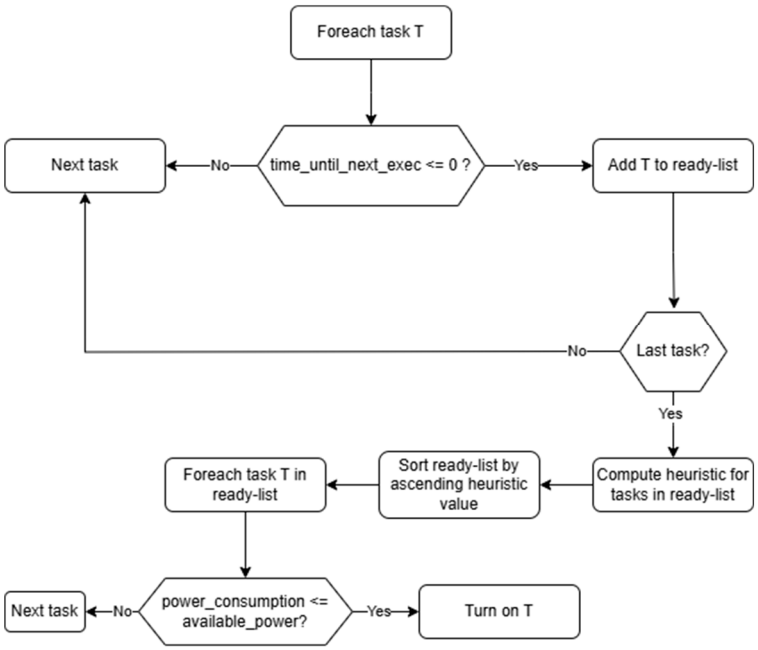



**Figure 1.** Scheduling flow.

By following the flow, it can be observed that it first loops through all tasks and checks whether a selected task T is ready for execution (time_until_next_exec <= 0). If it is not ready, it skips to the next task. Otherwise, add T to the ready list. It then keeps looping until all tasks have been checked. Once that is done, the scheduler computes a heuristic value for all tasks in the ready list. By only computing a heuristic for tasks in the ready list, the scheduler saves some computation time. Once the heuristics are computed, the ready list is sorted by ascending heuristic value. Since our algorithm aims to minimize the heuristic, a smaller value represents a higher overall priority—the reason for this design choice will be made apparent in Section 3.2.

After the sorting is done, the scheduler once again loops through all tasks in the ready list and checks whether there is enough available power to execute task T. If there is not,

skip the task and continue the loop. If there is, T is turned on, and the loop continues all the way to the point where the available power is not enough to execute any of the tasks, or all tasks from the ready list have been turned on. This effectively implements the knapsack technique used by several papers, as shown in the above section.

### *3.2. Problem Formulation*

The scheduling problem can be formulated as a constrained optimization problem. Let the following be true:

$x_i(t)$ : task activation decision.
$P_i$ : power consumption.
$E(t)$ : available power.
$D_i(t)$ : delay of task $i$.

The objective is to minimize task delays while respecting power constraints:

$$min \sum_i D_i(t) \cdot x_i(t)$$

Subject to the following constraints,

$$\sum_i P_i \cdot x_i(t) \leq E(t)$$
$$x_i(t) \in \{0, 1\}$$

### *3.3. Heuristic Function Design*

The heuristic function is the core component of our algorithm; thus, designing and tuning it is crucial to meet the defined requirements. It can be interpreted as a linear scalarization of a multi-objective optimization problem, with each component representing a different objective, and the weights defining their relative importance. By drawing inspiration from the previous works discussed in Section 2 as the main source of ideas, the respective advantages and disadvantages are evaluated in order to develop a flexible yet robust system.

The task definition outlined in Section 3.1 plays a key role in this process due to its large number of parameters for each task, which allows fine-grained adjustments to the heuristic and thus high level of flexibility to the user.

The heuristic function consists of six components:

1. Priority: Normalized difference between the maximum set priority (from ground crew) and the task's priority. It encodes mission-level importance.
2. Duration: Task duration, normalized by dividing it by its frequency (fixed value between 0 and 1 per iteration).
3. Delay: One of the most important components. It is the normalized difference between task frequency (in seconds) and time until the next execution. This value is by default negative, so it is negated. This component approximates urgency. This simple equation is quite powerful. This will be discussed in more detail later in this section.
4. Power: Task power consumption, divided by the expected power input (≈6 W for Aether [12]). This component enforces resource awareness.
5. Penalty for excess power: It takes the value 0 or M. The value is 0 if available power $\geq$ task consumption, otherwise M (a large value, which effectively denies the task from getting executed).
6. Bonus for non-interruptible active tasks: The opposite of the fifth component. It takes values of 0 or -M. -M ensures that non-interruptible tasks remain active.

The above components are then used in the following equation to compute the final heuristic value for a task:

$$H = w_{pr} \cdot Pr + w_{du} \cdot Du + w_{de} \cdot De + w_{po} \cdot Po + PE + BI \quad (1)$$

where $H$ represents the final heuristic value, $w_{pr}$ the weight of the priority component, $Pr$ the priority component, $w_{du}$ the weight of the duration component, $Du$ the duration component, $w_{de}$ the weight of the delay component, $De$ the delay component, $w_{po}$ the weight of the power component, $Po$ the power component, $PE$ excess power consumption penalty, and $BI$ the bonus for non-interruptible and already on.

The introduction of weights allows the mission designer to fine-tune the heuristic function's prioritization strategy by emphasizing specific components. As will be shown in Section 5, different weighting strategies can produce drastically different results.

Notice how the equation is mostly made of task parameters that do not change much after being set. The only exception is the delay component. When a new task is put into the queue or is reset from a previous cycle, a positive heuristic score will be assigned to it. With time passing and the time until the next execution coming closer, the delay component will decrease, pulling the final heuristic value closer and closer to zero. This ensures that a task will be put into a lower position, allowing it to make the cut and be executed.

In the event that the task fails to start up on time, that is, when there are still tasks with lower heuristic value taking up the knapsack, the delay component will continue decreasing into the negatives. As per Equation (1), this will induce a drastic change in the task's heuristic value that will lead to the task being considered for startup much sooner than any other task. Thus, as soon as enough power is available, the delayed task is scheduled for execution. To ensure that a delayed interruptible task does not incur more delay, the delay component keeps being decremented until the task finishes its execution. This makes it unlikely that the task is interrupted unless some critical situation arises.

Once the ready list is sorted, the task selector performs the knapsack technique by looping through all tasks in the ready list until either there is not enough power available to schedule any of the remaining tasks, or all the tasks are active.

Since the knapsack capacity is effectively limited by the available power, any drop in it will reduce the knapsack space, and thus reduce the amount of tasks that can be scheduled. This leads to an increase in the average execution delay. In certain situations, it could even lead to tasks with high power demands to be excluded entirely during low-power periods, further worsening the delays.

It is important to note that the scheduler operates based on instantaneous power availability and battery state of charge (SoC), without explicitly modeling or predicting future energy evolution across orbital phases such as lit/eclipse transitions. This design choice prioritizes real-time responsiveness and low computational cost. However, it limits the scheduler's ability to anticipate future energy constraints.

### *3.4. Adaptation to Flight Phases*

The scheduler operates across three flight phases: lit, eclipse and re-entry. The lit phase represents the time during which the satellite is exposed to sunlight. The eclipse phase represents the time during which the sun is blocked out by Earth. Finally, the re-entry phase represents the short time during which the satellite performs atmospheric re-entry. This phase is specific to the Aether mission as of the writing of this article. The first two are handled similarly, while in the re-entry phase, we only consider tasks for execution marked with an active_on_re-entry flag, which allows for selective activation and reduces the computation time.

Additionally, the scheduler has three distinct operating modes: normal, low-power, and critical. Each mode only allows certain tasks to be executed. This is necessary so that the ground crew can adjust thresholds to keep essential systems, such as the transmitter/receiver module, running even in critical situations, ensuring communication and allowing for manual intervention when needed. By setting a priority threshold that a task's priority level must meet for it to be scheduled, the ground crew can adjust which tasks run in which power mode. The mode selection depends on the battery's state of charge, with thresholds being customizable based on mission requirements. Table 2 shows the priority values associated with the different modes, with "100" being the maximum priority a task can have.

**Table 2.** Default values for different operating modes.

| | **Normal Mode** | **Low-Power Mode** | **Critical Mode** |
|---|---|---|---|
| Battery SoC range | 40–100% | 20–40% | 0–20% |
| Priority threshold | None | 50 | 80 |

### *3.5. Performance Metrics and Weight Optimization*

In order to be able to measure the performance of the custom scheduler, four performance metrics were established: maximum delay, average delay, average power consumption, and average execution time. The latter was less relevant as it was a fixed value per task set. With the help of these metrics, a clear picture could be established as to the performance of the scheduler.

As it was presented in Equation (1), each component of the final heuristic value can be weighted differently. This makes it possible for the scheduler to take different approaches to prioritizing tasks, which can dramatically influence the algorithm's performance, as described in detail in Section 5.

The default weighting used was a vector of ones. During various test scenarios (showcased in Section 5), it was found that this weighting does not always yield optimal scheduling results. To solve this problem, the weights were first manually varied in an attempt to achieve more optimal results by minimizing all values in the performance metrics described above. Manually adjusting weights, however, is very time consuming and inefficient.

To make the search more efficient, a custom search algorithm, akin to a stochastic hill climbing algorithm [13], was developed. This approach allowed for a more time-efficient and fully automated optimization. After an initial weight vector is fed to the search algorithm, the process starts. The objective was defined as lowering the maximum task execution delay in the performance metrics for at least three test scenarios out of five. If this was achieved, it was considered that improvement in three scenarios was worth the trade-off of potentially worse metrics in two scenarios compared to the initial setup, and thus a more optimal solution was found. The algorithm works by making random adjustments to the initial weight vector, running a simulation with those parameters, and comparing its results with the results yielded by the initial weight set. The search algorithm runs until either the objective is achieved, or it reaches the maximum number of iterations, set to be 100 in this case. This was done to prevent the search algorithm from attempting to look for a better solution in the "wrong direction" indefinitely, as well as having observed that after 100 iterations, any improvement was negligible.

Using this method, several solutions were found in a very short period of time, which drastically reduced the amount of time that had to be spent on optimization.

### *3.6. Conclusion*

As was shown in Section 3, every component of the scheduler is designed with flexibility in mind. Tasks can be updated or modified to suit the mission needs, while the operational modes allow adaptation to a large variety of mission conditions. Finally, the weight optimization mechanism allows the heuristic to be tailored to a vast number of mission profiles within the boundaries of the heuristic's design. Altogether, this design represents a flexible, lightweight and robust scheduling solution for CubeSats.

## 4. Simulation and Test Scenarios

This section presents the custom simulator developed alongside the scheduler, followed by the results obtained from various test scenarios.

### *4.1. Simulator*

To test and evaluate the performance of the custom scheduler, an in-house simulator suite was developed. It can be used to set up various test environments, by means of customizable parameters, and keeps track of different values (e.g., available power, flight phase, etc.) for the duration of the simulation.

As a real-time simulation would be a very time-consuming process, a 60× time compression factor is applied to each test case by default, reducing a 90 min orbit into 90 s one. Furthermore, the user also has the possibility to apply an extra time compression factor, to further reduce the simulation time. This must be done with caution as, after extensive testing, an additional time compression factor larger than five created non-negligible floating-point errors in the scheduler.

Additionally, it is important to note that the simulation environment is deterministic, and repeated runs lead to identical results. Therefore, statistical variability is not present. Future work will introduce stochastic disturbances to enable full robustness and variance analyses.

### *4.2. Test Scenarios*

To evaluate the performance of the custom scheduler, a number of test scenarios were created, each covering a specific edge case. Five benchmark scenarios were created to test how the scheduler functions in orbit only, without a re-entry phase. An additional set of five tests were also created specifically to test the scheduler's behavior during re-entry.

#### 4.2.1. In-Orbit Tests

Each test scenario presented in this section uses the following simulator parameters (Table 3):

**Table 3.** Simulator parameters for in-orbit tests.

| **Simulation Speed** | **5×** |
|---|---|
| Test time | 120 s |
| Lit time | 30 s |
| Eclipse time | 20 s |

A test time of 120 s was chosen, with a lit time of 30 s and an eclipse time of 20 s, which adds up to a 50 s orbit. Thus, in a 120 s-long test, the satellite completes close to 2.5 orbits. After meticulous testing, it was found that 120 s per test was an adequate compromise between stable results and waiting time.

The first test, Test 1, is designed to evaluate the performance of the scheduler when it is presented with a relatively small set of tasks that are made to overlap a fair bit in their

execution. This can be seen from the frequency values in Table A1. Each of these tasks has a distinct priority level, as well as a relatively high-power consumption. Indeed, throughout these tests, the input power was considered to be 6 W from the solar panels, as calculated by the Aether team [12]. Thus, when tasks overlap, there is a high chance that the power demand will exceed the power input at some point, resulting in the scheduler having to make a decision as to which task to prioritize.

Test 2 uses the exact same parameters as Test 1, but this time, the Interruptible parameter is set to False for all tasks. For this scenario, the goal was to observe whether the scheduler makes different scheduling decisions, seeing that once a task starts executing, it cannot be interrupted, and thus if a task is waiting for execution at the same time, a significant delay can accumulate.

Test 3 once again uses the same parameters as Test 1, but here the Interruptible parameter is set to True for all tasks. The goal for this scenario was to observe the decision-making process of the scheduler in a situation where any task can be interrupted at any time.

Test 4 differs from the previous three. Table A2 shows the scenario parameters. In this scenario, the scheduler is presented with a situation where it faces a large number of frequently repeating, short-duration, and low-power-consumption tasks, together with one scarcely repeating, long-duration, and high-power-consumption task. The goal was to determine whether the "large" task would be scheduled for execution at some point or hang indefinitely due to all the "small" tasks constantly taking priority over it.

Finally, Test 5, the parameters of which can be found in Table A3, was designed to simulate a realistic set of tasks with a payload task being the mission-critical component. In this scenario, the OBC is scheduled to run constantly throughout the duration of the simulation, thus constantly drawing a small amount of power.

#### 4.2.2. In-Orbit with Re-Entry Tests

For this set of tests, the following simulation parameters were used (Table 4):

**Table 4.** Simulator parameters for in-orbit and re-entry tests.

| **Simulation Speed** | **1×** |
|---|---|
| Test time | 120 s |
| Lit time | 30 s |
| Eclipse time | 20 s |
| Time until re-entry | 100 s |
| Re-entry duration | 20 s |

Once again, a test time of 120 s is chosen. This time, however, the satellite is deorbited after two orbits and then spends 20 s in the re-entry phase. During that time, the solar panels are inoperable, and the satellite must rely solely on its batteries for power.

The tests presented in this section will all be based on the parameters found in Table A4. There, it can be seen that the OBC is once again set to run for the entire duration of the test. There is also a heat shield task, scheduled to start its execution when the satellite begins re-entry. This task is performed in order to test the CubeSat re-entry technology. Should the heat shield task not be able to execute for any reason, the mission is considered to have been a catastrophic failure, resulting in the loss of the vehicle.

While all five tests presented here use the same set of tasks and parameters from Table A4, each test differs in the starting conditions. These are presented in the following Table 5:

**Table 5.** Starting conditions for the in-orbit and re-entry tests.

| | RTest 1 | RTest 2 | RTest 3 | RTest 4 | RTest 5 |
|---|---|---|---|---|---|
| Input power | 6 W | 3 W | 6 W | 4 W | 4.5 W |
| Initial battery charge | 85% | 85% | 85% | 85% | 22% |
| Battery capacity | 22.5 Wh | 22.5 Wh | 15 Wh | 15 Wh | 15 Wh |

The initial conditions presented in the table above represent the following cases:

RTest 1 represents a nominal situation, where the satellite's solar panels and battery function correctly.

RTest 2 represents a situation where the solar panels and battery cannot output the initially anticipated amount of power, thus leading to fewer tasks being able to execute at the same time and a longer charge time for the battery.

RTest 3 shows a situation where the battery capacity got lowered due to, for example, excessive battery degradation.

RTest 4 presents a situation where both the solar panels and the battery degraded, resulting in a suboptimal power input and power capacity.

Finally, RTest 5 represents a situation where both the solar panels and battery degraded, and the battery was initially at a much lower state of charge, making the satellite operate in low-power mode. This situation is used to assess whether the battery would survive re-entry without completely depleting.

## 5. Results

The shown results are based on the performance metrics as defined in Section 3.4, for each test scenario. Additionally, five sets of weights were selected for the scheduler, with simulations repeated for each, and the results were compiled and compared on a single graph for each performance metric.

Table 6 shows the sets of weights used by the scheduler. The benchmark set is used to establish a standard to which the performance of other weight sets can then be compared. Pr10 and Po10 are weight sets found through manual testing, whereas Opt3 and Opt4 are weight sets found by the weight optimizer algorithm, as presented in Section 3.5.

**Table 6.** Weight sets used during testing.

| | Priority | Duration | Delay | Power |
|---|---|---|---|---|
| Benchmark | 1 | 1 | 1 | 1 |
| Pr10 | 10 | 1 | 1 | 1 |
| Po10 | 1 | 1 | 1 | 10 |
| Opt3 | 6.88 | 3.95 | 4.21 | −9.38 |
| Opt4 | 4.99 | 3.65 | 0.73 | −10.92 |

### *5.1. Maximum Delay*

The maximum delay is the most critical metric. It is directly correlated with operational reliability and thus is the most important metric to optimize for. The results are shown in the Figure 2.

The benchmark set shows a maximum delay of more than 40 s in Test 4, which is nearly a full 50 s orbit. This is caused by the large, non-interruptible task being delayed by multiple smaller interruptible tasks. Test 3 also shows a significant delay.

Pr10 significantly improved performance, cutting Test 4's maximum delay by nearly 80% and showing small gains in Tests 3 and 5, with Tests 1 and 2 unchanged. Po10 had the opposite effect: prioritizing high-power tasks more than doubled Test 4's delay to 99.58 s

(2 full orbits) and significantly increased the delay in Test 3, despite slight improvements in Test 5.

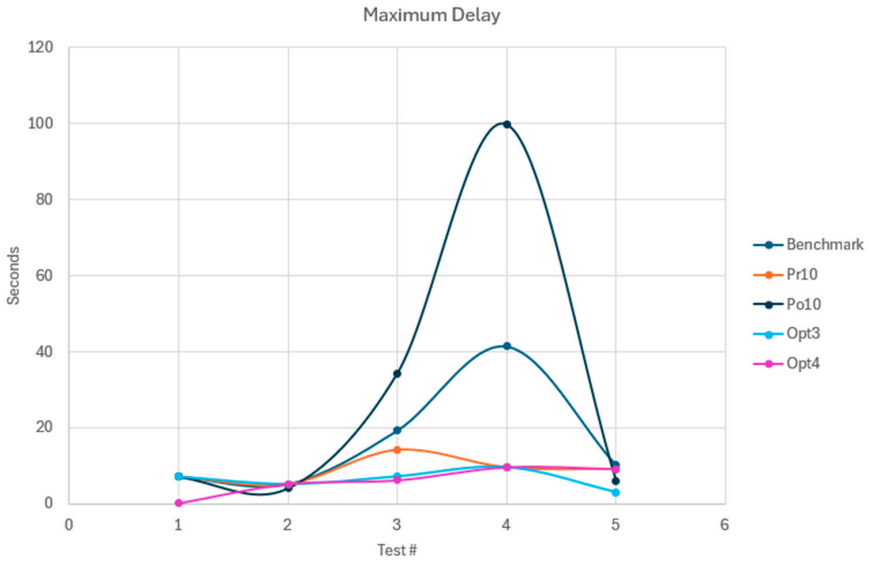



**Figure 2.** Maximum task delay per test case.

Opt3 matched Pr10 in Tests 1, 2, and 4, but cut the delay by 50% in Test 3 and by more than 66% in Test 5, yielding the lowest delay observed overall. Opt4 mirrored Opt3's pattern: it drastically reduced Test 1's delay to 0.1 s, matched delays in Tests 2 and 4, and slightly improved Test 3, but had barely any effect in Test 5.

While Opt4 significantly improved Test 1 over the benchmark, delays in Test 2 stayed, overall, constant. Tests 3 and 4 showed wide variability, while Test 5 remained relatively stable, with the exception of the improvement brought by Opt3.

### *5.2. Average Delay*

The average delay is a running mean of delays for all tasks during the simulation. It is another key metric impacting operational reliability alongside the maximum delay.

Three distinct patterns can be seen in Figure 3. First, the benchmark and Pr10 curves are nearly identical, with only minor differences. While Pr10 greatly improves the maximum delay (Section 5.1), it slightly increases the average delay in most scenarios.

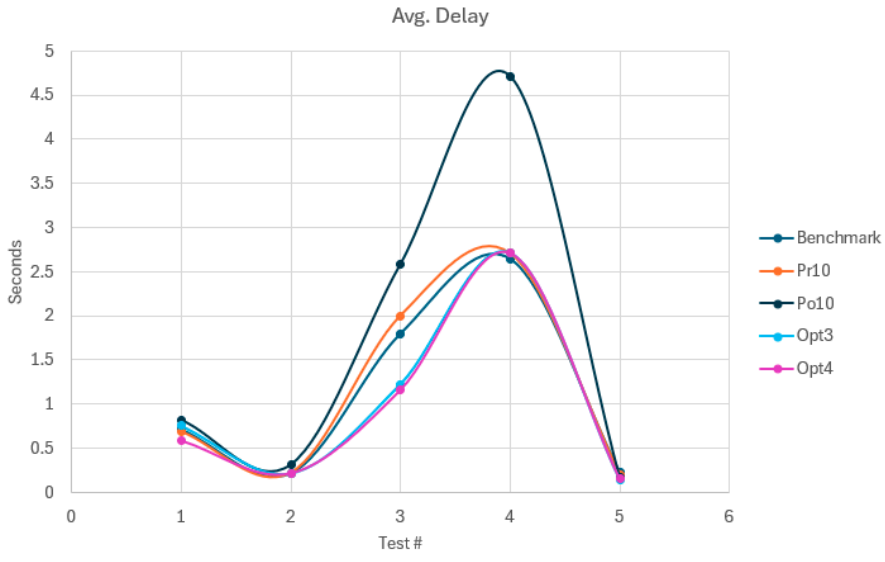



**Figure 3.** Average task delay per test case.

Second, Opt3 and Opt4 closely match each other, showing minor improvements in Test 1 and especially in Test 3, where the average delay dropped by around 0.5 s compared to the benchmark. Tests 2, 4, and 5 showed no significant changes and remained close to the benchmark values.

Finally, Po10's behavior closely resembles that of the maximum delay, performing worse than the benchmark in most scenarios. The largest impact appears in Tests 3 and 4, with the average delay in Test 4 rising by over 40%.

### *5.3. Average Power Consumption*

The average power consumption, like the average delay, is a rolling mean computed over the course of the simulation. The results are shown in the Figure 4.

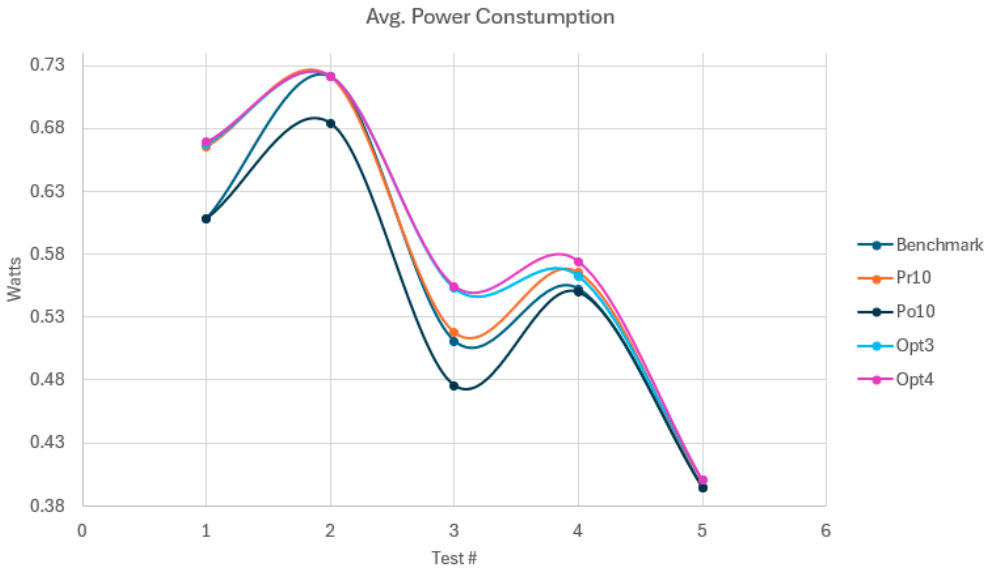



**Figure 4.** Average power consumption per test case.

The benchmark shows the second-lowest consumption of all weight vectors, with only Po10 achieving lower values. While this may seem favorable, the delays associated with Po10 are significant (Sections 5.1 and 5.2). Furthermore, the difference in the average power consumption between the various weight sets is minimal, about 0.075 W at most, which makes the trade-off between slightly lower consumption and increased delays generally unattractive.

Pr10 closely follows the benchmark except in Test 1, where consumption rises by around 0.6 W. Opt3 and Opt4 show similar trends but with increases in Tests 1, 3, and 4, and a small rise in Test 5.

Po10, which prioritizes high-power tasks, achieves lower average consumption in four of five tests and matches the benchmark in one. This potential benefit explains its inclusion in this paper, though it comes at the cost of sharply reduced operational reliability under most test conditions. The acceptability of this trade-off will likely depend on specific mission requirements.

### *5.4. Timeplot of Test 4, Po10 Configuration*

To showcase how tasks behave during Test 4, with the Po10 weight configuration, the timeplot in Figure 5 was created. The plot shows that the nine small tasks overwhelm the scheduler, causing the single large task to hang indefinitely and thus to never execute. Thus, while the scheduler aims to optimize task execution, the user must give careful consideration to the number, execution window, and power consumption of tasks, to avoid situations like this.

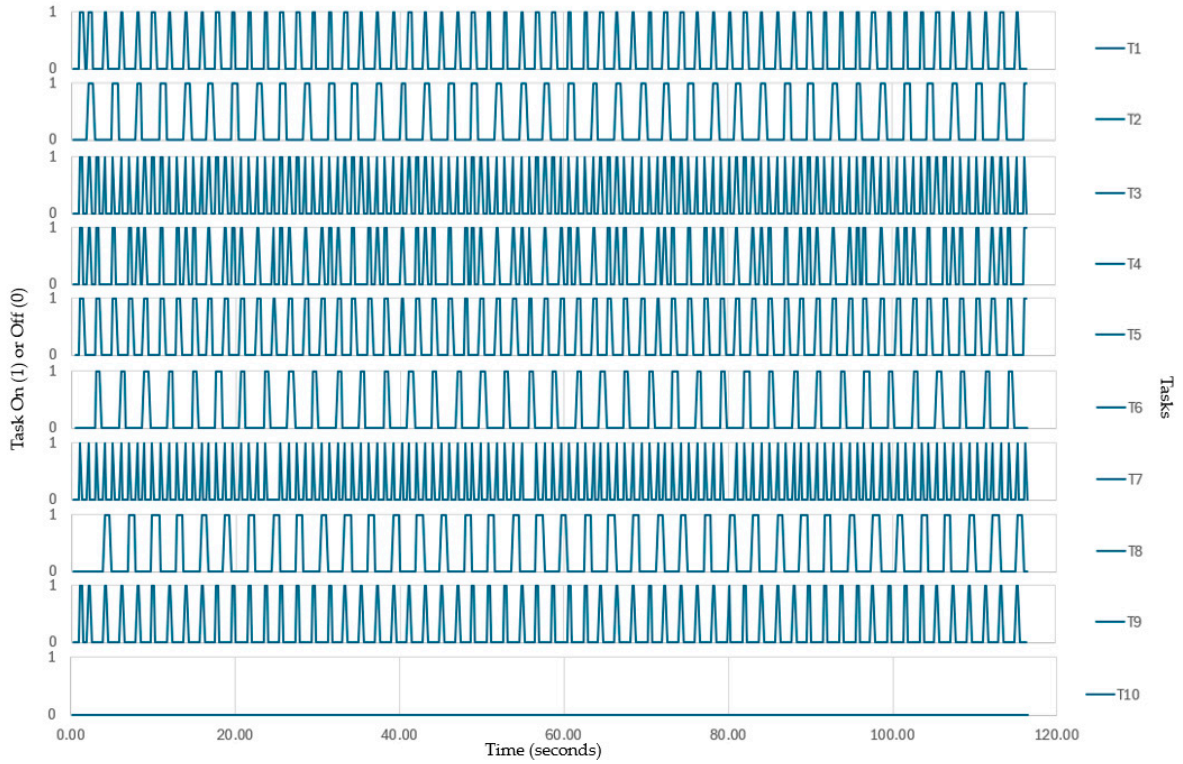



**Figure 5.** Timeplot of Test 4, with Po10 weight configuration.

### *5.5. In-Orbit and Re-Entry Results*

The goal for these tests was to evaluate how different starting conditions, showcased in Table 5, will affect the outcome of the mission. Most importantly, it is determined whether, at any point, the mission would suffer a catastrophic failure that would result in the loss of the vehicle. The following table showcases the results of these tests.

It can be observed in Figure 6 that in every scenario, the scheduler manages to maintain and even increase the battery's state of charge over the course of the simulation. This clearly indicates that even under circumstances where the satellite suffers from various failures, the scheduler manages to execute tasks while maintaining enough power reserves for the satellite to safely complete re-entry.

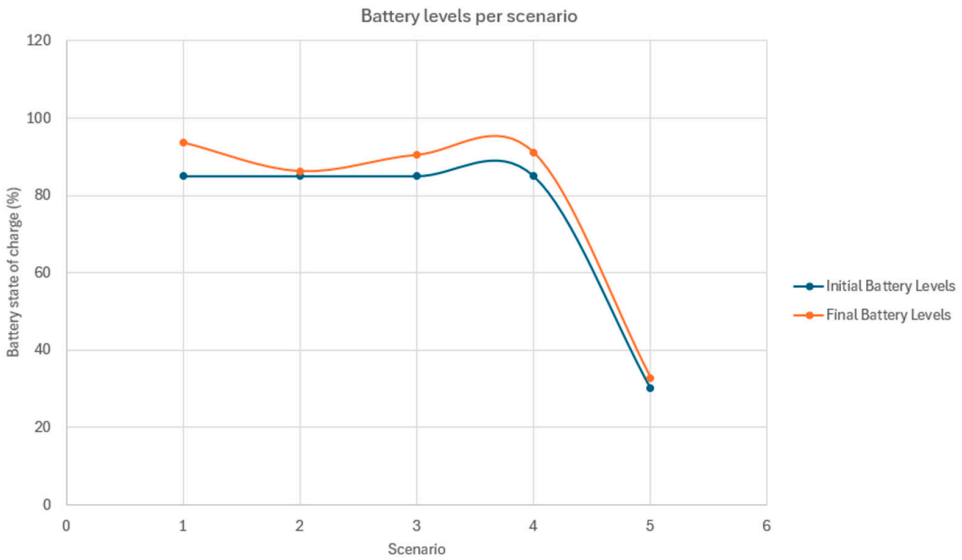



**Figure 6.** Initial and final battery levels per test case.

## 6. Discussion

In this work, a heuristic-based scheduling and power management algorithm was developed. It ensures operational reliability under a variety of flight conditions and adapts to changing conditions in real time. The scheduler ranks tasks using a weighted heuristic and selects tasks for execution by means of a knapsack method, allowing for adaptive scheduling in changing flight conditions. The development and use of a stochastic hill-climbing-like search algorithm showed that there exist weight configurations that yield far better results than others in various test cases. These configurations, coupled with the scheduler's three operating modes, demonstrated that significant gains can be made compared to the benchmark in terms of operational reliability at a relatively low cost when it comes to overall power consumption. Additionally, the algorithm's modular architecture enables straightforward integration into a wide range of mission profiles. These results demonstrate that carefully tuned heuristic scheduling can substantially enhance operational reliability at the cost of a slight increase in the overall power consumption and can provide a scalable solution for future CubeSat missions.

The results specifically highlight several important trade-offs that are inherent to this scheduling approach. First, minimizing task delay generally leads to a slight increase in power consumption, as more tasks are executed earlier or concurrently. However, this increase remains relatively small, in the order of 0.075 W, which indicates that substantial improvements in scheduling reliability can be achieved at a relatively low energy cost. At the same time, increasing the weight of the task priority improves the execution of critical tasks but may reduce fairness, potentially leading to starvation of lower-priority tasks in highly constrained scenarios, as demonstrated in Test 4. Furthermore, the increase in the weight of the power component reduces the overall energy consumption but significantly degrades the scheduling performance, as higher-power tasks are less likely to be selected for execution. This shows that overly conservative power-aware strategies can negatively impact the overall mission performance. Overall, the results show that the scheduler is highly versatile and vastly different scheduling strategies can be achieved through weight tuning. Consequently, the proposed framework showcases a scheduling solution that focuses on flexibility rather than a single optimal solution, allowing mission designers to tailor the scheduling strategy to meet the specific operational requirements. Nevertheless, care must be taken to choose appropriate task frequencies during mission design, to avoid ending up in sub-optimal scheduling scenarios such as excess power draw.

Future work could include adding predictive capabilities to the scheduler to further enhance its adaptability. Another point of improvement could be to expand the scheduler to be able to accommodate dependent or compound tasks, as currently it can only work with independent tasks. Furthermore, expanding upon the scheduler by considering more EPS metrics can bring a significant improvement to the robustness, reliability and energy optimization that the scheduler can offer. Such metrics may include depth of discharge, deep discharge limit and power optimization indicators such as MPPT effectiveness.

**Author Contributions:** Conceptualization, M.K. and D.A.; software, M.K. and D.A.; validation, M.K.; investigation, M.K. and D.A.; resources, V.D.S.; data curation, M.K.; writing—original draft preparation, M.K. and D.A.; writing—review and editing, M.K., D.A., V.D.S. and J.V.; visualization, M.K.; supervision, V.D.S. and J.V. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by KU Leuven IOF.

**Data Availability Statement:** The source code presented in this study can be obtained upon request from the corresponding authors.

**Acknowledgments:** The authors express their gratitude to the University of KU Leuven for its support and encouragement.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Appendix A

**Table A1.** Test 1 parameters.

| | Task 1 | Task 2 | Task 3 | Task 4 | Task 5 |
|---|---|---|---|---|---|
| Duration (s) | 3 | 6 | 2 | 2 | 8 |
| Frequency (s) | 6 | 20 | 8 | 12 | 18 |
| Power consumption (W) | 2 | 1 | 3 | 2 | 4 |
| Interruptible | True | False | True | True | False |
| Priority | 70 | 50 | 30 | 80 | 40 |
| On status | False | False | False | False | False |
| Active on re-entry | False | False | False | False | False |

**Table A2.** Test 4 parameters.

| | Task 1 | Task 2 | Task 3 | Task 4 | Task 5 | Task 6 | Task 7 | Task 8 | Task 9 | Task 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Duration (s) | 0.5 | 0.8 | 0.3 | 0.4 | 0.6 | 0.7 | 0.2 | 0.9 | 0.4 | 10 |
| Frequency (s) | 0.6 | 0.9 | 0.4 | 0.5 | 0.7 | 0.8 | 0.3 | 1 | 0.5 | 20 |
| Power consumption (W) | 0.8 | 1 | 0.5 | 0.7 | 0.9 | 1.2 | 0.6 | 1.1 | 0.8 | 5.8 |
| Interruptible | True | True | True | True | True | True | True | True | True | False |
| Priority | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 100 |
| On status | False | False | False | False | False | False | False | False | False | False |
| Active on re-entry | False | False | False | False | False | False | False | False | False | False |

**Table A3.** Test 5 parameters.

| | OBC | ADCS | Heaters | Misc Task 1 | Misc Task 2 | Misc Task 3 | Payload |
|---|---|---|---|---|---|---|---|
| Duration (s) | Test duration | 0.5 | 0.25 | 5 | 10 | 2 | 10 |
| Frequency (s) | Test duration | 10 | 2 | 15 | 30 | 8 | Lit time + eclipse time |
| Power consumption (W) | 0.5 | 1 | 1.5 | 1 | 0.7 | 1.6 | 4 |
| Interruptible | False | True | True | True | True | True | False |
| Priority | 100 | 70 | 50 | 30 | 30 | 30 | 100 |
| On status | True | False | False | False | False | False | False |
| Active on re-entry | False | False | False | False | False | False | False |

**Table A4.** In-orbit and re-entry test parameters.

| | OBC | ADCS | Heaters | Misc Task 1 | Comms | Heat Shield | Payload |
|---|---|---|---|---|---|---|---|
| Duration (s) | Test duration | 0.5 | 0.25 | 5 | 4 | 60 | 10 |
| Frequency (s) | Test duration | 5 | 2 | 15 | 12 | Time until re-entry | Lit time + eclipse time |
| Power consumption (W) | 0.5 | 1 | 1.5 | 1 | 1 | 2 | 4 |
| Interruptible | False | False | True | True | True | False | False |
| Priority | 100 | 90 | 50 | 30 | 30 | 100 | 100 |
| On status | True | False | False | False | False | False | False |
| Active on re-entry | True | True | True | False | True | True | False |

## References


1. Kay, L. *How Do Prizes Induce Innovation? Learning from the Google Lunar X-Prize*; Georgia Institute of Technology: Atlanta, GA, USA, 2011.
2. Baker, A.; Phipps, A.; Davies, P.; Alabart, X.; Sweeting, M. Smallsats to the Moon: Providing the 'Picks and Shovels for the 21st century's Greatest Exploration Endeavour'. 2007. Available online: https://digitalcommons.usu.edu/smallsat/2007/all2007/7/ (accessed on 9 March 2026).
3. Hussain, B.; Guo, J.; Fareed, S.; Uddin, S. Robotics for Space Exploration: From Mars Rovers to Lunar Missions. *Int. J. Ethical AI Appl.* **2025**, *1*, 1–10. [CrossRef]
4. Nicks, O.W. *Far Travelers: The Exploring Machines*; Scientific and Technical Information Branch, National Aeronautics and Space Administration: Washington, DC, USA, 1985.
5. Poghosyan, A.; Golkar, A. CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions. *Prog. Aerosp. Sci.* **2017**, *88*, 59–83. [CrossRef]
6. Aether. Design of an Algorithm for a Re-Entry CubeSat That Optimizes Power Consumption and Controls Subsystem Priorities in Different States. Internal Technical Document. Available online: https://aetherspace.be/public/rexus (accessed on 9 March 2026).
7. Thakurta, V.; Datla, V.; Ravi, A.; Reddy, R.; Jain, A.; Akhoury, A.; Parashar, A.; Dali, H.; Kejriwal, A. Design and Implementation of Power Management Algorithm for a Nano-satellite. In *Proceedings of the 2019 IEEE Aerospace Conference, Big Sky, MT, USA, 2–9 March 2019*; IEEE: Piscataway, NJ, USA, 2019.
8. Martinez, S.V.; Seman, L.O.; Filho, E.M.; Slongo, L.K.; Bezerra, E.A. On-board energy scheduling optimization algorithm for nanosatellites. *Int. J. Circuit Theory Appl.* **2023**, *51*, 3915–3937. [CrossRef]
9. Bernardo, V.P.; Seman, L.O.; Bezerra, E.A.; Ribeiro, B.F. Hardware in the Loop Simulation of an On Board Energy Driven Scheduling Algorithm for CubeSats. *IEEE Embed. Syst. Lett.* **2023**, *16*, 69–72. [CrossRef]
10. Rigo, C.A.; Seman, L.O.; Camponogara, E.; Filho, E.M.; Bezerra, E.A. A nanosatellite task scheduling framework to improve mission value using fuzzy constraints. *Expert Syst. Appl.* **2021**, *175*, 114784. [CrossRef]
11. Rigo, C.A.; Seman, L.O.; Camponogara, E.; Filho, E.M.; Bezerra, E.A. Task scheduling for optimal power management and quality-of-service assurance in CubeSats. *Acta Astronaut.* **2021**, *179*, 550–560. [CrossRef]
12. Aether. Solar Cells Calculations. Internal Technical Document. Available online: https://aetherspace.be/ (accessed on 9 March 2026).
13. Selman, B. Hill-climbing Search. *Encycl. Cogn. Sci.* **2006**, *81*, 10.


**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.