Artifact for RTSS 2025: Probabilistic Response-Time-Aware Search for Transient Astrophysical Phenomena.

This repository contains the source code, datasets, and analysis tools supporting the RTSS 2025 paper, *"Probabilistic Response-Time-Aware Search for Transient Astrophysical Phenomena."*

It is available at https://github.com/Daisy0419/Telescope-Searching-Problem/tree/rtss2025_artifact

Table of Contents

System Requirements

- OS: Linux (some instructions are Ubuntu-specific)

- CPU: Minimum 2 cores, **8+ cores preferred** (24 physical/48 logical cores used in paper)
- RAM: **8GB**
- Required Storage, Option A, Docker Container: **Total: 5.6GB**
  - Docker package install: 500MB
  - Docker container: 5.1GB
- Required Storage, Option B, Local Install: **Total: 4.1GB**
  - Repo: 200MB
  - Miniconda: 800MB
  - Conda environments: 2.8GB
  - Gurobi: 200MB
  - LEMON: 70MB

Overview

This artifact addresses the problem of scheduling follow-up observations of astrophysical transients under real-time constraints. The task is to select and order sky tiles for telescope observations to maximize the expected figure of merit (FoM) associated with successful detection of an afterglow within one or more hard deadlines and given dwell-time and slew-time constraints.

Included Components

- C++ implementations of all algorithms: GCP, ILP, Genetic, Greedy
- Precomputed results in CSV format for:
  - Small and large problem instances under a single deadline
  - Small problem instances incorporating maximum-observed execution time (MOET)
  - Large problem instances under multi-deadline settings
- A Jupyter notebook to reproduce plots and figures from the paper
- Long-running Python scripts to recompute results from scratch

Directory Structure

```
.
├── include/                          # C++ header files
├── src/                              # C++ source implementations
├── CMakeLists.txt                    # CMake build configuration

├── skytiling/
│   ├── precompute_tile_maps.sh     # Precomputes tile boundaries by projecting telescope FoV
│   ├── flatten_healpix.py          # Convert multi-order hierarchical HEALPix
│   │                                 to flat HEALPix format
│   ├── visualize_healpix.py        # Generate images of HEALPix maps
│   ├── produce_tilings.py          # Creates all tilings
│   └── produce_single_tiling.py    # Generates tiling from one HEALPix map for one telescope

├── data/                           # Tiling CSV files for small-scale sky maps
│   ├── small/                      # Tiling CSV files for small-scale sky maps
│   ├── large/                      # Tiling CSV files for large-scale sky maps
│   ├── large_moet/                 # Subset of large-scale maps used in MOET-aware experiments
│   ├── ligo_healpix/               # HEALPix likelihood maps from LIGO
│   ├── ligo_healpix_flattened/     # Flattened HEALPix maps
│   ├── ligo_healpix_images/        # Image (.png) representations of the HEALPix maps
│   └── moet/                       # Maximum-observed execution time measurements

├── results/
│
│   ├── figures/                     # Figures 7–10 in paper
│
│   ├── precomputed_results/         # Precomputed outputs used in the paper
│   │   ├── small/                   # FoM results for small instances
│   │   ├── large/                   # FoM results for large instances
│   │   ├── small_with_moet/         # MOET-aware FoM for small instances
│   │   └── multi_deadline/          # Multi-deadline scenario results
│
│   ├── recomputed_results/          # Outputs from re-running experiments
│   │   ├── small/
│   │   ├── large/
│   │   ├── small_with_moet/
│   │   └── multi_deadline/
│
│   ├── visualize_results.ipynb        # Notebook for Figures 7–10 in paper
│   ├── run_small_instances.py         # Batch run script for small instances
```

```
│   ├── run_large_instances.py         # Batch run script for large instances
│   ├── run_small_instances_moet.py    # Batch run script (MOET-aware)
│   ├── run_multi_deadline.py          # Batch run script for multi-deadline setup
│   └── run_all.sh                     # Runs all four scripts sequentially

├── rtss25-sky-tiling.yml               # Python dependencies for tiling code
├── rtss25-telescope-search.yml         # Python dependencies
└── README.md
```

1 Environment Setup

You can run the artifact via **Docker (recommended)** or a **Local Setup**. A Gurobi license is needed only to run ILP-based experiments.

1.1 (Preliminary, Optional) Obtaining a Gurobi License

Our algorithms include an ILP implementation of the orienteering problem, which is solved using the commercially-available Gurobi Optimizer.

Gurobi requires a license (`gurobi.lic`) to run.

- Refer to: How to Retrieve and Set Up a Gurobi License
- **Note:** If you do not have a Gurobi license, you can still run experiments 3.2, 3.3, and 3.4 below, which do not invoke the ILP solver.

If you are an academic user, Gurobi provides **free academic licenses**:

- Free Academic License
- **Note**: If you're using an academic license and intend to run the experiments using our provided Docker container, be sure to request an Academic WLS License (floating license). Named-User Academic Licenses are not compatible with Docker containers.

To obtain an academic license:

1. Navigate to the Gurobi portal. https://portal.gurobi.com/
2. Login or register to create a free Gurobi account using your academic email address.
3. Navigate to Gurobi's academic license request page. https://portal.gurobi.com/iam/licenses/request/?type=academic
4. Under "WLS Academic," click, "Generate Now!"

5. Download the generated `gurobi.lic` file.
6. Move or copy the file to the path of your choice. All commands listed hereafter assume it is in `~/gurobi.lic`.

1.2 (Option A, Preferred) Using the Provided Docker Container

1.2.1 Install Docker

You may install Docker according to these instructions. Here, we include the instructions for Ubuntu distributions:

1. Set up Docker's `apt` repository:

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] \
  https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

2. Install the latest Docker packages.

```
sudo apt-get install docker-ce docker-ce-cli \
  containerd.io docker-buildx-plugin docker-compose-plugin
```

1.2.2 Pull the Docker Image

```
sudo docker pull ghcr.io/daisy0419/rtss25-op-solver:1.0
```

All dependencies are pre-installed, and project binaries are precompiled in the image. You can jump to Reproducing Paper Figures or Running Full Experiments.

1.3 (Option B) Local Installation

1.3.1 Clone Repository

Clone the repository to the path of your choice. All commands listed hereafter assume it is placed directly into your home directory.

```
cd ~
git clone -b rtss2025_artifact https://github.com/Daisy0419/Telescope-Searching-Problem/
```

1.3.2 Python Environment Setup

We recommend setting up a conda environment for Python.

If you do not have conda installed locally:

```
cd ~/Telescope-Search-Problem
```

Download and install Miniconda (change ~/conda to your preferred location)

```
export CONDA_DIR=~/conda
wget -q https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O miniconda.sh
bash miniconda.sh -b -p "${CONDA_DIR}"
rm miniconda.sh
```

Make conda available in your shell

```
. "${CONDA_DIR}/etc/profile.d/conda.sh"
conda config --system --set channel_priority flexible
```

Once conda is available, create the python environments using provided yaml files:

```
conda env create -f rtss25-sky-tiling.yml
conda env create -f rtss25-telescope-search.yml
```

Activate **rtss25-sky-tiling** before running tiling scripts or **rtss25-telescope-search** before running all other scripts:

```
conda activate rtss25-sky-tiling
# or
conda activate rtss25-telescope-search
```

1.3.3 C++ Environment Setup

(1) Gurobi Optimizer (Required)

The Gurobi Optimizer is used to solve our ILP approach to the orienteering problem.

1. Download and extract Gurobi to the directory of your choice. All commands listed hereafter assume it is placed directly in your home directory.

```
cd ~
wget https://packages.gurobi.com/12.0/gurobi12.0.3_linux64.tar.gz
tar xvfz gurobi12.0.3_linux64.tar.gz
```

2. Set the necessary environment variables in your shell (change `~/gurobi1203` to your preferred location).

```
export GUROBI_HOME=~/gurobi1203/linux64
export PATH="${GUROBI_HOME}/bin:$PATH"
export LD_LIBRARY_PATH="${GUROBI_HOME}/lib:$LD_LIBRARY_PATH"
```

Note: If you don't have Gurobi license and you do not plan to run experiments involving Gurobi, you still need to install Gurobi in order to compile the project code (due to build-time linking requirements).

(2) LEMON Graph Library (Required)

The Lemon Graph Library is used to compute the minimum-weight perfect matching used in our Greedy Christofides Pathfinding algorithm.

1. Download, extract, and build the LEMON Graph Library to the directory of your choice. All commands listed hereafter assume it is placed directly in your home directory.

```
wget http://lemon.cs.elte.hu/pub/sources/lemon-1.3.1.tar.gz
tar xvfz lemon-1.3.1.tar.gz
cd lemon-1.3.1
mkdir build && cd build
cmake ..
make -j
sudo make install
```

2. Set the necessary environment variables in your shell (change `~/lemon-1.3.1` to your preferred location).

```
export LEMON_SOURCE_DIR=~/lemon-1.3.1
export LEMON_BUILD_DIR=~/lemon-1.3.1/build
```

(3) Build the C++ Executables

Once all dependencies are installed, you can build the C++ project with:

```
cd ~/Telescope-Search-Problem
mkdir build && cd build
cmake ..
make -j
```

This produces two binaries in `build/`. The only difference between them is the `main` function:

- **ts** — built from `src/main.cpp`. Entry point for batch experiments used in the paper, invoked by the Python scripts in `results/`.
- **op** — built from `src/main_custom.cpp`. Entry point for single-case runs, one algorithm on one (skymap, budget, slewrate) instance.

---

2 Reproducing Paper Figures

You can visualize the results via Jupyter notebook either via **container** or **locally**.

2.1 Run Jupyter Notebook via Docker Container

```
sudo docker run --rm -it -p 8888:8888 \
  -v "$PWD:/workspace" \
  ghcr.io/daisy0419/rtss25-op-solver:1.0 \
  bash -lc 'conda run -n rtss25-telescope-search \
    jupyter lab --ip=0.0.0.0 --port=8888 --no-browser \
      --IdentityProvider.token="" \
      --ServerApp.root_dir=/workspace \
      --allow-root'
```

Then open http://localhost:8888 and navigate to **results/visualize_results.ipynb** in the sidebar.

2.2 Run Jupyter notebook Locally

```
conda activate rtss25-telescope-search
cd ~/Telescope-Searching-Problem/results
jupyter notebook visualize_results.ipynb
```

2.3 Reproducing Results in the Jupyter Notebook

The notebook includes:

- Average percentage deviation from ILP baseline across deadlines for small instances(Fig.7)
- Percentage deviation in FoM from the GCP baseline and Computation time vs Deadline presented in log-scaled plot for large instances (Fig. 8)
- FoM percentage deviation from GCP after accounting for MOET. for five small instances (Fig 9)
- Expected FoM progression over the multi deadlines (Fig. 10)

All required `.csv` results are precomputed and stored in `results/precomputed_results`.

The notebook saves figures to `results/figures`.

---

3 Running Full Experiments

To re-run the full set of experiments (~6 hours total runtime even on a machine with the recommended specs), you may either set up the environment locally or use the provided Docker container. Experiments can then be executed using the provided batch scripts.

3.1 Set Up the Run Environment

3.1.1 Run Experiment in a Docker Container

If you have a Gurobi license on your local machine, mount the license into the container.

License Note: If you're using an academic license, be sure to request an Academic WLS License (floating license). Named-User Academic Licenses are not compatible with Docker containers.

```
cd ~
sudo docker run  -p 8888:8888 --rm -it -v "./gurobi.lic:/licenses/gurobi.lic:ro" \
 -e GRB_LICENSE_FILE=/licenses/gurobi.lic ghcr.io/daisy0419/rtss25-op-solver:1.0
```

If you do not have a Gurobi license, you can still run experiments that do not rely on ILP-based solvers:

```
sudo docker run -p 8888:8888 --rm -it ghcr.io/daisy0419/rtss25-op-solver:1.0
```

3.1.2 Run Experiment Locally

```
cd ~/Telescope-Searching-Problem
```

3.2 Regenerate Sky Tilings (Data Preparation, Optional)

The sky tiles used for the small and large problem instances evaluated are already stored in the `data/small` and `data/large` directories.
Optionally, you may regenerate them.
All commands in this step are run from the `sky_tiling` directory.

```
cd ~/Telescope-Search-Problem/sky_tiling
```

Note that much of the tiling code is modified from the Python 2.7 project at https://github.com/shaonghosh/sky_tiling

3.2.1 Precompute FoV Projections (~ 10 minutes)

Each telescope's FoV is projected onto the unit sphere to precompute tile boundaries.

```
conda activate rtss25-sky-tiling
bash precompute_tile_maps.sh
```

This step is optional.
Files already exist in `sky_tiling/tile_center_files` and `sky_tiling/tile_pixel_maps`.

Recomputed files appear in those same directories, with file names containing `recomputed`.

3.2.2 Flatten HealPix (~ 10 seconds)

Multi-order hierarchical HEALPix maps in `data/ligo_healpix`
need to be flattened into a non-hierarchical format.

```
conda activate rtss25-telescope-search
python flatten_healpix.py
```

This step is optional.
Files already exist in `data/ligo_healpix_flattened`;
they will be overwritten by this script.

3.2.3 Visualize HealPix (~ 15 seconds)

Just for visualization purposes, we generate PNG image
representations of the flattened HEALPix maps.

```
conda activate rtss25-telescope-search
python visualize_healpix.py
```

This step is optional.
Files already exist in `data/ligo_healpix_images`;
they will be overwritten by this script.

3.2.4 Generate Tiles (~ 15 seconds)

The flattened HEALPix maps and precomputed tilings are intersected
to produce the tiles, with probabilities, that serve as the inputs to the search problem.

```
conda activate rtss25-sky-tiling
python produce_tilings.py
```

Files in `data/small` and `data/large` will be overwritten.

3.3 Rerun all Experiments

Each Python script corresponds to a different experiment setting.
If you do not have a Gurobi license, you can still run experiments 3.3.2, 3.3.3, and 3.3.4, which do not rely on ILP solvers.

All commands in this step are run from the `results` directory and use the default conda environment.

```
conda activate rtss25-telescope-search
cd ~/Telescope-Search-Problem/results
```

Running Time

Running this complete set of experiments takes ~6 hours even on a powerful machine with the recommended system requirements. We therefore provide two options for convenience:

(Option 1) Run all experiments from a single script

Experiments 3.3.1-3.3.4 may be run sequentially by running the following script:

```
bash run_all.sh
```

(Option 2) Limit the coverage

You may modify the scripts to use fewer problem instances and to test fewer deadlines.

The first line of each script is:

```
run_all = True
```

Change this to `False` to run a limited problem set. The implications for each script are listed under each experiment below.

3.3.1 Small Instances (~ 2 hours)

```
python3 run_small_instances.py
```

Results will be saved to `results/recompute_results/small`.

If `run_all = False` then only two small instances, with deadlines 10 and 90, will be run. **(< 1 minute)**

3.3.2 Large Instances (~ 3 hours)

```
python3 run_large_instances.py
```

Results will be saved to `results/recomputed_results/large`.

If `run_all = False` then only two large instances, with deadlines 10, 100, and 1000, will be run. **(~ 1 minute)**

3.3.3 Small Instances with MOET (~ 40 minutes)

```
python3 run_small_instances_moet.py
```

Results will be saved to `results/recomputed_results/instances_with_moet`.

This script evaluates algorithms with budgets that account for MOET (Maximum Observed Execution Time). It runs in **three parts**:

(1) Collect raw runs to estimate MOET

For each skymap and each budget in {10,20,…,100}, the script runs Greedy, Genetic, and GCP 5 times each using the same original budget (no MOET applied yet). Results are stored in `results/recomputed_results/instances_with_moet/get_moet/`.

(2) Compute MOET per (Skymap, Method, Budget)

Across the 5 runs in (1), it then computes:

```
MOET(Skymap, Method, Budget) = max(TimeSec)
```

and writes the results to `results/recomputed_results/instances_with_moet/moet/<skymap>.csv`

(3) Re-run with MOET-adjusted budgets

For each (Skymap, Method, Budget), the budget is reduced by the corresponding MOET:

```
adjusted_budget = max(0, original_budget - MOET(Skymap, Method, original_budget))
```

It then runs the instances again with these adjusted budgets and saves the results to `results/recomputed_results/instances_with_moet/result_with_moet/out_<skymap>.csv`

If `run_all = False` then only two small instances, with deadlines 10 and 100, will be run. The MOET will only be computed across a single run of each. **(~ 1 minute)**

3.3.4 Multi-Deadline Large Instances (~ 1 minute)

```
python3 run_multi_deadlines.py
```

Results will be saved to `results/recompute_results/multi_deadlines`.

If `run_all = False` then only two large instances will be run. **(~ 10 seconds)**

3.4. Visualizing the Results

Again, you can visualize the results via Jupyter notebook either **inside the container** or **locally**.

- **Inside the container**

```
conda activate rtss25-telescope-search
jupyter notebook --ip=0.0.0.0  --port=8888 \
  --no-browser --allow-root --IdentityProvider.token=""
```

Open http://localhost:8888 in a browser and navigate to **results/visualize_results.ipynb**.

- **Locally**

```
conda activate rtss25-telescope-search
cd ~/Telescope-Searching-Problem/results
jupyter notebook visualize_results.ipynb
```

All required `.csv` results are stored in `results/recomputed_results`.

---

4 Extensibility of Experiments

4.1 Running a Single Algorithm with Designated Tiling, Speed, and Time Budget

We provide a command-line interface (CLI) to run **one algorithm at a time** on a given problem instance. As before, a Gurobi license is needed only to run the ILP-based algorithm.

Usage

```
cd build
./op <file> <budget> <alg> [slew_rate=50]
```

Arguments

<file>: path to a CSV file specifying the set of tiles, with probabilities, for the problem instance (see Input format below).

<budget>: total time budget (slew + dwell).

<alg>: one of greedy | genetic | gcp | ilp.

[slew_rate] (optional): slew rate in degrees per unit time (default 50 degrees per second).

Examples

Greedy on a small instance, budget = 50, default slew_rate

```
./op ../data/small/filtered_GW191105_143521_7dt_separate.csv 50 greedy
```

Genetic on a large instance with a larger budget and custom slew_rate

```
./op ../data/large/filtered_GW191103_012549_7dt.csv 500 genetic 30
```

GCP on a large instance

```
./op ../data/large/filtered_GW191109_010717_7dt.csv 200 gcp
```

ILP (Gurobi) on a small instance

```
./op ../data/small/filtered_GW191105_143521_7dt_separate.csv 50 ilp 40
```

Input Format

The input is a CSV tiling file with the following first five columns in order (additional columns after these, e.g. CumulativeProb, are ignored):

(1) Rank — rank of filtered tiles (integer)

(2) index — tile ID in the original tiling (integer; not used but kept for index alignment)

(3) RA — right ascension of the tile center in degrees [0, 360)

(4) Dec — declination of the tile center in degrees [-90, 90]

(5) Probability — probability that the TAP originated in this tile (i.e., prize)

Header example:

```
Rank,index,RA,Dec,Probability[, ...]
```

Output

The result will be printed to stdout:

- Path (node sequence)
- Number of nodes
- Sum probability (FoM)
- Wall-clock runtime (seconds)

4.2 Modify Dwell Times

For now, the dwell time calculation, based on air mass as described in Section IX of our paper, is hard-coded. To modify this, you will need to modify `src/ReadData.cpp` then re-build.

4.2.1 Modify reference time

All dwell times use 1 second as a normalized baseline; this is then scaled according to the air mass corresponding to the telescope's pointing direction.

```
src/ReadData.cpp:

 190 | double dwelltime_ref = 1.0;
```

4.2.2 Modify scaling function

The air mass is computed in the function `computeAirMass`, which is then used to compute the relative scaling of the irradiance in `computeScalingFactor`, and then applied to the dwell time in `computeDwellTime`.

```
src/ReadData.cpp:

199 | double airMass = computeAirMass(zenithAngle);
200 | double scalingFactor = computeScalingFactor(airMass);
201 | dwell_times[i] = computeDwellTime(dwelltime_ref, scalingFactor);
```

All of these functions are present in `src/ReadData.cpp` and can be modified as needed. For example, if you want to use a constant dwell time for each tile, you could simply modify as:

```
src/ReadData.cpp:

200 | double dwell_time = 100;
201 | dwell_times[i] = dwell_time;
```

4.2.3 Rebuild

- **Inside the container**

```
cd build
make -j
```

- **Locally**
  Follow the steps under **1.3.3 C++ Environment Setup** above.

4.3 Add Custom Algorithm

Here, we describe how to add your own algorithm for solving the s-t Orienteering Problem.

All algorithms have a wrapper function that share the following signature:

```
std::vector<int> my_algo(const std::vector<std::vector<double>>& costs,
                         const std::vector<double>& prize,
                         double budget, int s, int t);
```

- `costs` is a reference to a complete, symmetric matrix of edge costs, i.e., `costs[i][j]` is the movement cost from vertex `i` to `j`.
- `prize` is a reference to an array of prizes, corresponding to tile probabilities.
- `budget` is the time deadline for the search, i.e., the deadline.
- `s` and `t` are the indices of the starting and ending vertices.

4.3.1 Write Your Algorithm

1. Create source and header files for your algorithm in `src/` and `include/`.
2. Write your algorithm, making sure it conforms to the above signature.

4.3.2 Call Your Algorithm

Your algorithm will be called from `src/main_custom.cpp`.

1. Modify `src/main_custom.cpp` to `#include` your header file.
2. Add a dispatch branch in the conditional logic to call your algorithm.

```
92 | if (alg_lc == "greedy") {
93 |   (void)run_and_report("Greedy", [&]{
94 |     return prizeGreedyPathTwoFixed(costs, probability, eff_budget, start_idx, end_idx);
95 |   }, costs, probability, ranks, padding);
96 | }
97 | else if (alg_lc == "genetic") {
98 |   (void)run_and_report("Genetic", [&]{
99 |     return genetic_optimization_st(costs, probability, eff_budget, start_idx, end_idx);
100 |   }, costs, probability, ranks, padding);
101 | }
      /***
      **** Your algorithm here!
      ***/
102 | else if (alg_lc == "myalgo") {
103 |   (void)run_and_report("MyAlgo", [&]{
104 |     return my_algo(costs, probability, eff_budget, start_idx, end_idx);
105 |   }, costs, probability, ranks, padding);
106 | }
```

4.3.3 Rebuild

1. If running **Locally** (i.e., **not** inside the Docker comtainer), set Gurobi environment variables:

```
export GUROBI_HOME=~/gurobi1203/linux64
export PATH="${GUROBI_HOME}/bin:$PATH"
export LD_LIBRARY_PATH="${GUROBI_HOME}/lib:$LD_LIBRARY_PATH"
```

2. Rebuild:

```
cd build
cmake ..
make -j
```

3. Run with your algorithm according to the details in **4.1 Running a Single Algorithm** above:

```
./op <file> <budget> myalgo [slew_rate=50]
```

4.4 Create Tilings for Different Telescope FoVs

The above instructions detail how to test our algorithms with different configurations for telescope dwell times and slew speeds. You may also generate new tilings for telescopes with different (square) fields of view.

4.4.1 Enter directory and activate conda environment

```
cd ~/Telescope-Searching-Problem/sky_tiling
conda activate rtss25-sky-tiling
```

4.4.2 Precompute tile boundaries by projecting telescope's FoV onto unit sphere

Each telescope's FoV is projected onto the unit sphere to precompute tile boundaries.

Pick a name of your choice to replace `YourTelescopeName` and specify the fov in degrees (e.g., 30):

```
python setup.py --telescope "YourTelescopeName" --fov 30
```

New files will be produced in `sky_tiling/tile_center_files` and `sky_tiling/tile_pixel_maps`.

4.4.3 Generate tiles from HealPIX map

A flattened HEALPix map and precomputed tilings are intersected
to produce the tiles, with probabilities, that serve as the inputs to the search problem.

1. Modify the first 5 lines of `produce_single_tiling.py`:

```
name = "GW200216_220804" # Name of the GW event for which you would like to make tiles
telescope = "7dt" # Replace this with the name you selected for the telescope
confidence_interval = 0.99 # Include highest-probability tiles until
                           # cumulative probability exceeds this value
max_rows = None # Maximum number of tiles to include, None means no limit
```

2. Run the script:

```
conda activate rtss25-sky-tiling
python produce_single_tiling.py
```

The tiling will be generated as `data/custom/{name}_{telescope}.csv`.