IAC-21,A6,7,9,x63767

# SENSIT: a software suite for observation scheduling and performance assessment of SST sensor networks

**Giovanni Purpura[a]*, Andrea De Vittori[a], Riccardo Cipollone[a], Pierluigi Di Lizia[a], Mauro Massari[a], Camilla Colombo[a], Alessandra Di Cecco[b], Luca Salotti[b]**

[a] *Politecnico di Milano, Via Giuseppe La Masa, 34, 20156 Milano, Italy*, giovanni.purpura@polimi.it, andrea.devittori@polimi.it, riccardo.cipollone@polimi.it, pierluigi.dilizia@polimi.it, mauro.massari@polimi.it, camilla.colombo@polimi.it

[b] *Italian Space Agency (ASI), via del Politecnico snc, 00133 Rome, Italy*, alessandra.dicecco@asi.it, luca.salotti@asi.it

* Corresponding Author

**Abstract**

The ability to simulate the behavior of different sensor configurations is critical for the development of a sensor network that provides data for Space Surveillance and Tracking (SST) services. Any software suite devoted to this shall be able to assess the performance of existing networks in terms of effectiveness and robustness, as well as to estimate the effects of structural changes, such as the addition or the upgrade of sensors. This paper is devoted to describing how SΞNSIT tackles the above problem. SΞNSIT (Space Surveillance Sensor Network SImulation Tool) is a software suite designed to perform an analysis of the observational and cataloging capabilities of a sensor network. The software can model optical, radar and laser ranging sensors and simulate different operational scenarios. The user shall define the sensors composing the network and a population of space objects. Typical sensor properties that can be set include type, mode (survey/tracking), location, accuracy, pointing constraints, detectability limits and operating hours. Inputs are processed to predict transits that can be observed by each sensor. This allows to assess the network capabilities in terms of catalog coverage: the sensors are compared against each other to identify overlapping in the sets of observable objects and estimate the level of complementarity or redundancy. Afterwards, the tool can simulate the operations of the network. First, an observation schedule is compiled, using a genetic optimization algorithm based on tunable criteria. This removes overlaps caused by objects passing simultaneously in the field of regard. Then, the software simulates and processes the measurements gathered during passes, carrying out orbit determination, aiming at assessing the network capability in terms of catalog build-up and maintenance. The results are illustrated in tables and graphs with different levels of detail, starting from a general performance overview up to the list of the passes. The user can also browse the object catalog of the network and analyze its evolution. Moreover, the tool allows to export intermediate data, such as the observable passes, the optimized schedule, and the pointing requirements. The modularity of the software grants easy modification of the properties of the network, to carry out a sensitivity analysis to different parameters. This is expected to ease the setup process of sensor networks for SST, as well as the identification of the most promising upgrades to be recommended. The paper presents in detail the software architecture and its functionalities, and shows the results provided in typical use cases.

**Keywords:** Sensor networks; Space object cataloguing; Space Surveillance and Tracking; Space Situational Awareness; Genetic algorithm

**Acronyms/Abbreviations**

Comma-Separated Values (CSV); European Space Agency (ESA); Field Of Regard (FOR); Field Of View (FOV); Graphical User Interface (GUI); Initial Orbit Determination (IOD); JavaScript Object Notation (JSON); Local Orbital Frame (QSW); Non-Linear Least Squares (NLS); Orbit Determination (OD); Radar Cross Section (RCS); Refined Orbit Determination (ROD); Space Surveillance and Tracking (SST); Two-Line Element set (TLE)

## 1. Introduction

The space environment has become a valuable asset for communication, navigation and observation purposes over the past years. Since 1957, more than 4900 space launches have led to an orbital population of more than 23000 trackable objects with sizes larger than 10 cm [1].

About a thousand of these are operational satellites, while the remaining 94% are space debris – objects that no longer serve any useful purpose. About 64% of the routinely tracked objects are fragments from some 250 breakups, mainly explosions and collisions of satellites or rocket bodies. In addition, about 670000 objects larger than 1 cm and 170 million objects larger than 1 mm are

expected to be in orbit. A schematic representation of the entire space debris population is given in *Fig. 1*.

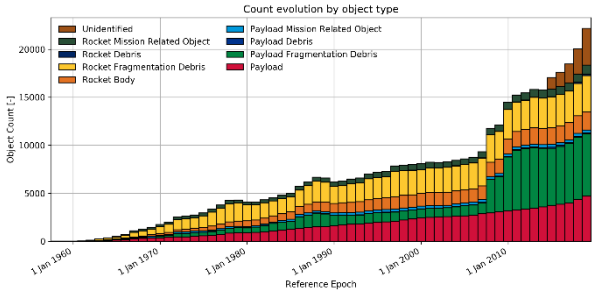


*Fig. 1 Timeline of the number of space debris in orbit (plot available on the ESA website* [2]*).*

Due to the increasing number of satellites, potential collisions with other objects and uncontrolled debris reentry that may endanger populated areas are of major concern. To mitigate these risks, surveying and tracking such objects is becoming of primary importance, as well as providing this information to a variety of stakeholders. The amount of catalogued objects in orbit scales with the quality of the available space surveillance systems. Hence, simulating a sensor network can have a significant impact when dealing with catalogue build-up and maintenance.

At European level, two examples of available sensor network simulation tools are the BAS3E (Banc d'Analyse et de Simulation d'un Systeme de Surveillance de l'Espace - Simulation and Analysis Bench for Space Surveillance System) and the S3TOC (Spanish Space Surveillance and Tracking Operations Center).

BAS3E is a complete SST simulation framework developed by CNES [3], with the goal to evolve the existing SST network, both from a software and hardware point of view, and to define major evolutions of existing SST networks. It implements the capability to simulate ground and space-based sensors via the integration of the following functions:

- Detection, tracking and generation of observations of space objects
- Object identification and tracking correlation
- Orbit determination
- Maintenance of a space debris catalogue
- Centralized / de-centralized tasking and scheduling

The S3TOC is located in the Torrejón de Ardoz Military Air Base [4], 30 km away from Madrid (Spain). The center is devoted to the generation of SST end-user products, for which a catalogue of objects is maintained, and orbital information from SST observations obtained by the S3TSN (Spanish Space Surveillance and Tracking Sensor Network) is computed. The S3TOC consists of the following elements:

- Data Processing and Cataloguing
- Service Processing
- Sensor Planning and Tasking
- Fragmentation messages
- Service Provision

The Italian SΞNSIT software provides functionality similar to those of its European counterparts. SΞNSIT is a tool for modeling sensor networks and evaluating their performance in terms of coverage and capability of building and maintaining a catalogue of space objects. Moreover, it allows the user to perform sensitivity analysis of the performance of the sensor network by varying the network configuration.

## 2. SΞNSIT

*Fig. 2 SΞNSIT logo.*

The Space Surveillance Sensor Network Simulation Tool (SΞNSIT) is a software tool conceived by Politecnico di Milano in collaboration with the Italian Space Agency and with contributions from the SpaceDyS company. The first version of this software had already been presented previously [5]; this work introduces the most recent features and reviews the main characteristics that were already present.

SΞNSIT is written in Python 3 and C++ and runs on the major operating systems (Windows, MacOS, Linux). It relies on the NASA/NAIF SPICE library [6] for astronomical computations. It can be used either from the command line or by means of a Graphical User Interface (GUI) based on the Qt library.

The software makes use of YAML files for the configuration and an SQLite database file for internal data storage.

Given a list of space objects, a sensor network and a time frame, SΞNSIT performs the following tasks:

- computation of the observable transits of space objects over the selected ground stations
- creation of optimal schedules of observations, according to user-defined criteria
- simulation of the observations and the corresponding measurements
- orbit determination using the simulated measurements and the provided sensor accuracies
- catalogue build-up and maintenance according to the outcomes of the orbit determinations

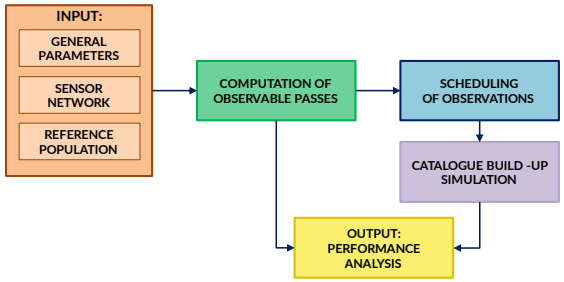


*Fig. 3 SΞNSIT architecture.*

These tasks are performed by the five modules into which SΞNSIT is subdivided (*Fig. 3*):

- **Data initialization**: gathers and pre-processes the inputs provided by the user and updates the SPICE kernels (files containing planetary ephemerides and leap second information).
- **Pass computation**: evaluates the observable passes of the objects belonging to the reference population, accounting for various observability constrains.
- **Scheduling of observations**: selects a subset of the previously computed passes (or parts of them) to produce optimal schedules of observations by means of a genetic algorithm.
- **Catalogue build-up**: starting from a schedule, simulates the measurements and carries out orbit determination, in order to build-up and maintain the network catalogue as the simulation proceeds.
- **Performance analysis**: allows to analyze the data created by the previous modules, by means of tables and charts, providing an overview of the performance of the sensor network.

## *2.1. Data initialization*

The data initialization module allows to manage the configuration of the software, to define the sensor network and to load the reference population of objects to be observed.

### *2.1.1. Configuration*

In the configuration tab of the GUI (*Fig. 4*) the user can define the following properties:

- generic parameters to tune the processes,
- accuracy thresholds, that will be used to determine if an object is considered catalogued according to the covariance of its position,
- time windows for the simulation,
- Space-Track credentials for automatic download of TLEs (optional).

The GUI saves these parameters in specific files in YAML format. If desired, the user can directly edit these files.

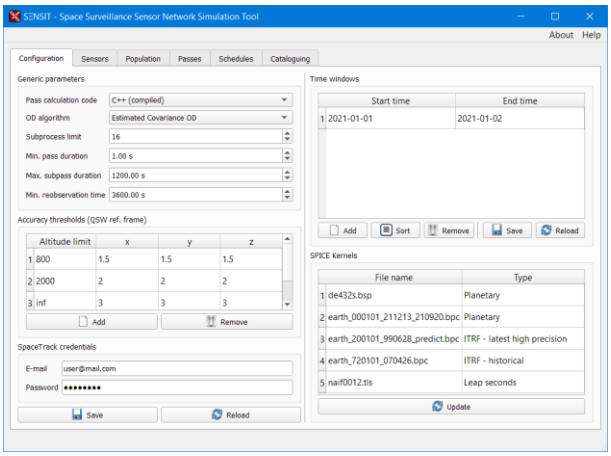


*Fig. 4 SΞNSIT GUI: configuration tab.*

### *2.1.2. Sensors*

The user shall configure the sensor network either through the GUI (*Fig. 5*) or by means of a YAML file formatted according to the instructions reported in the software manual.

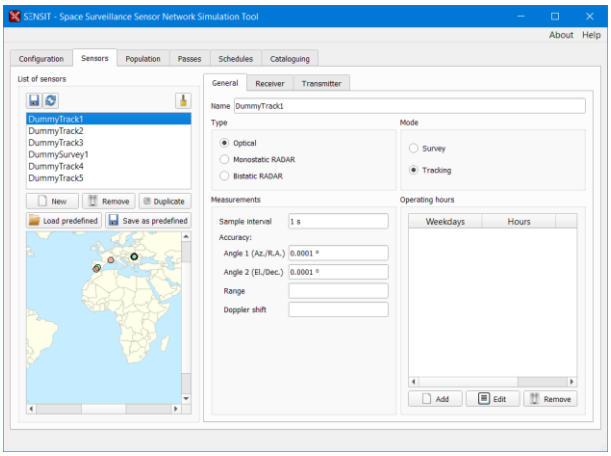


*Fig. 5 SΞNSIT GUI: sensors tab – general.*

The parameters to be entered for each sensor are:

- Name
- Type (optical, radar mono/bistatic)
- Mode (tracking, survey)
- Working hours (optional)
- Measurement sample interval
- Measurement accuracies

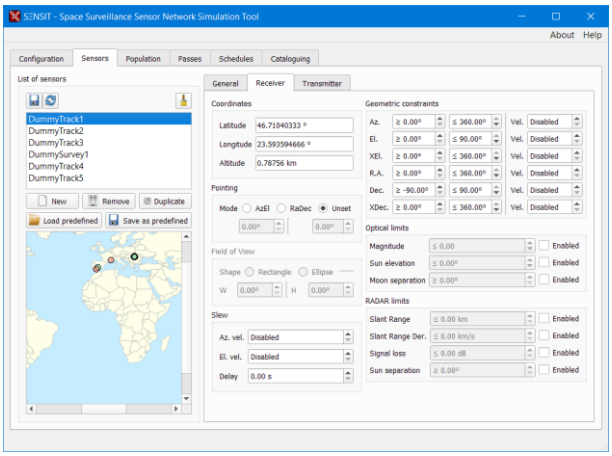


*Fig. 6 SΞNSIT GUI: sensors tab – receiver.*

For the receiving station (*Fig. 6*) it is necessary to enter:

- Geographical coordinates
- Pointing and Field of View (for survey sensors)
- Slew speed (for tracking sensors)
- Geometrical constraints (e.g., Field of Regard)
- Optical and radar signal limits

If the sensor is a bistatic radar, the user shall also provide the information about the transmitting station. In this case, the fields are the same as those for the receiver, except for the optical and radar signal limits that are not present.

### *2.1.3. Reference population*

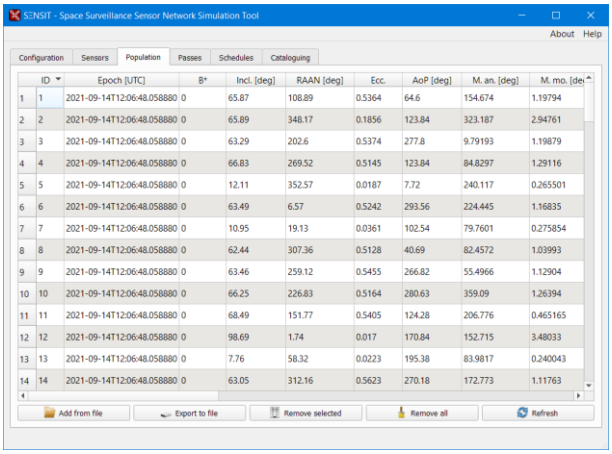


*Fig. 7 SΞNSIT GUI: population tab.*

The reference population of space objects can be loaded through the GUI (*Fig. 7*) or with a command line script, using the following formats:

- Two-Line Elements in a text file,
- Cartesian states in CSV format,
- List of Satellite Catalog Numbers (NORAD IDs), for which TLEs will be automatically downloaded from Space-Track.org,
- ESA MASTER population file (*.pop),
- SGP4 elements (in CSV or JSON format).

The inputs are automatically converted to SGP4 elements when necessary and are saved in the internal SQLite database.

The user can also associate an initial state covariance for the objects expressed in one of the typically used reference frames [7].

Furthermore, it is possible to load the Radar Cross Section and the intrinsic brightness of the objects: these are used to compute the radar signal loss and the optical magnitude, respectively.

The signal loss is computed according to *Eq. 1*.

$$L_{dB} = 20\log_{10}(\rho_{RX}\cdot\rho_{TX}) - 10\log_{10}\sigma + 30\log_{10}(4\pi) - \log_{10}c \qquad (1)$$

where $\rho_{RX}$ and $\rho_{TX}$ are, respectively, the distance of the target from the receiver and the transmitter in meters, $\sigma$ the RCS in squared meters and $c$ the speed of light in meters per second. Considering the link budget equation, the value of signal loss limit to use can be determined from the characteristics of the sensor, as in *Eq. 2*.

$$max.\ L_{dB} = 10\log_{10}(min.P_{RX}) - 10\log_{10}P_{TX} + 10\log_{10}G_{RX} + 10\log_{10}G_{TX} - 10\log_{10}f \qquad (2)$$

with $min.P_{RX}$ minimum detectable received power in watt, $P_{TX}$ transmitted power in watt, $G_{RX}$ and $G_{TX}$ receiver and transmitter gains, $f$ carrier frequency in hertz.

Optical magnitude is computed according to *Eq. 3*.

$$m = b - 2.5\log_{10}((\pi-\phi)\cos(\phi) + \sin(\phi)) + 5\log_{10}\rho - 15 - e \qquad (3)$$

with $b$ intrinsic brightness, $\phi$ phase angle in radians, $\rho$ distance from the target in km and $e$ atmospheric extinction, computed with *Eq. 4* [8].

$$e = \frac{0.1451e^{-h/7.996} + 0.120e^{-h/1.5} + 0.016}{\sin(el) + 0.025e^{-11\sin(el)}} \qquad (4)$$

with $h$ altitude of the ground station in km and $el$ elevation angle in radians.

### *2.2. Pass computation*

This process computes the observable passes of the objects belonging to the reference population, considering the observability conditions set by the user. The computed passes are shown in the GUI (*Fig. 8*).

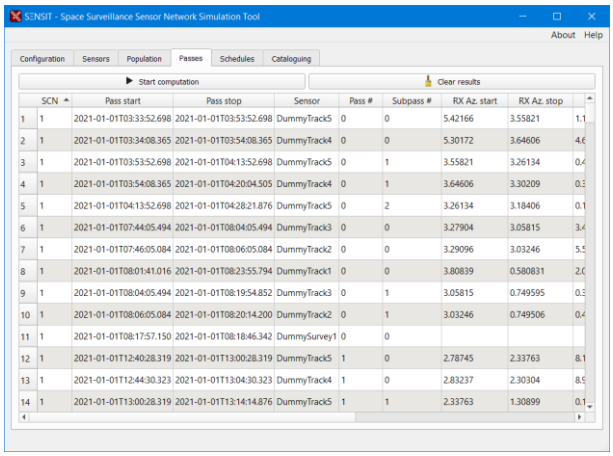


*Fig. 8 SΞNSIT GUI: passes tab.*

The computation is based on a bisection algorithm and takes advantage from several optimizations:

- The states of the stations and of the objects are cached when possible
- The core of the algorithm is written in C++
- The code runs in parallel on separate processes

The algorithm, schematized in *Fig. 9*, starts from the time window chosen by the user and splits it into segments. At each splitting point, it evaluates the observability condition, giving a preliminary estimation of the passes. At the end, it applies bisection to find the precise start and stop epochs of the passes.

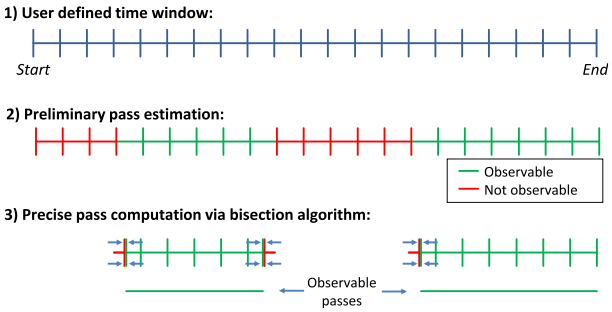


*Fig. 9 SΞNSIT algorithm for passes computation.*

## *2.3. Scheduling of observations*

Several objects may be observable by each sensor at the same time. For this reason, it is necessary to schedule the observations for the sensors that are in tracking mode. The section of SΞNSIT that addresses this problem is shown in *Fig. 10*.

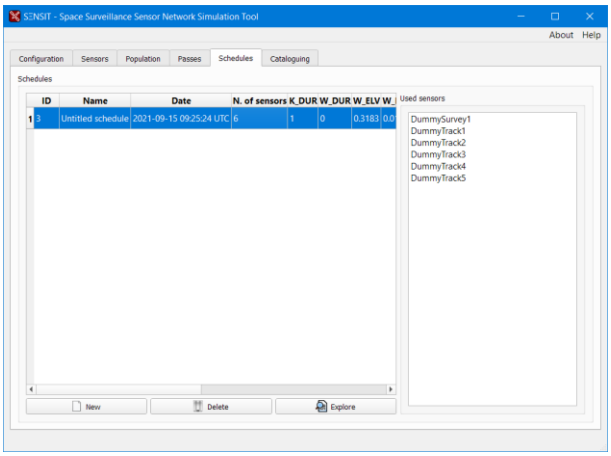


*Fig. 10 SΞNSIT GUI: schedules tab.*

### *2.3.1. Scheduling rules*

The scheduling rules that have been implemented in SΞNSIT are:

- For survey sensors, all observable passes can be observed.
- For tracking sensors, a pass can be observed when:
  - the sensor is not currently busy in the observation of another object,
  - the sensor has the time to perform the slew maneuver from a pass to the next,
  - a given time has passed since the last observation of this object.

When two passes cannot be present in a schedule at the same time without violating the scheduling rules, they are in conflict. Given an ordered list of $n$ observable passes $\boldsymbol{p}$, the conflicts may be summarized as a $(n, n)$ Boolean matrix $\mathbf{M}$: if the $i^{th}$ pass conflicts with the $j^{th}$, the matrix will contain *true* both in cell $(i, j)$ and in cell $(j, i)$. Conversely for passes that are not in conflict the corresponding cells will be set to *false*. The conflict matrix has the following properties:

- It is symmetric by definition,
- Its diagonal values are all *false*, since a pass never conflicts with itself,
- It is in general sparse (i.e., most of its values are *false*) and can therefore be stored in memory-efficient representations.

Even the schedule can be represented using a Boolean vector $\boldsymbol{s}$ associated to the ordered list of passes. If the $i^{th}$ pass is to be observed, the $i^{th}$ item of the vector will be *true*, otherwise it will be *false*.

*Fig. 11* shows an example of passes that overlap: if we consider for simplicity only the constraint that a sensor can observe only one object at a time, pass #1 conflicts with pass #0 and pass #2, while pass #3 has no conflicts. This will be encoded in the matrix $\mathbf{M}$ reported in *Fig. 12*, where *true* is reported in the positions (0, 1)

and (1, 0). A possible schedule **s** without conflicts can be composed of pass #0, #1 and #3, as in *Fig. 12*.

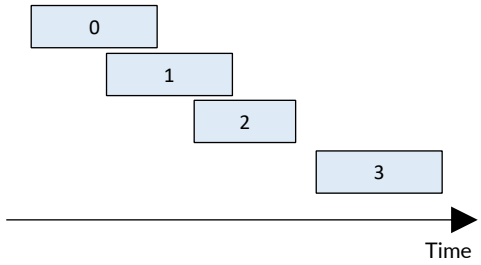


*Fig. 11 Example of a timeline of passes with overlaps.*

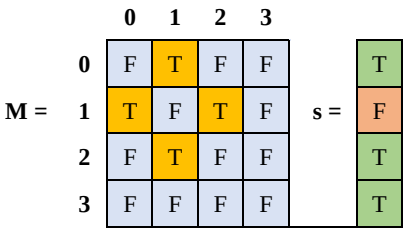


*Fig. 12 Boolean representation of a conflict matrix (**M**) and a schedule (**s**).*

This representation allows to easily compute the number of conflicts *c* present in a schedule using *Eq. 5*, where Boolean values *true* and *false* are implicitly converted to the integers 1 and 0.

$$c = \frac{\mathbf{s}^{\mathbf{T}}\mathbf{M}\,\mathbf{s}}{2} \qquad (5)$$

This is possible because the product **M s** returns a vector where the i[th] value is the number of other chosen passes in conflict with this pass. The next pre-multiplication by $\mathbf{s}^{\mathbf{T}}$ will sum only the values corresponding to chosen passes. The division by 2 is necessary to count each conflict only once.

In the situation depicted in *Fig. 12*, *Eq. 5* returns 0 as expected. If instead all the passes had been scheduled, the formula would have returned 2, because of the two conflicts caused by pass #0 with pass #1 and #2.

### *2.3.2. Fitness value*

The set of all possible schedules is composed of $2^n$ elements, of which only the ones that fulfil the condition $c = 0$ are feasible. In order to programmatically prefer one schedule over the others, it is necessary to assign a *fitness value* to each of them.

First, a score $f_i$ is computed for each observable pass (*Eq. 6*), then the overall *fitness F* of the schedule is computed by summation of the scores of the scheduled passes (*Eq. 7*).

$$f_i = {v_d}^{k_d} \sum_n w_n v_n \text{ , with n} \in (d, e, \rho, L, m) \qquad (6)$$

$$F = \boldsymbol{s} \cdot \boldsymbol{f} \qquad (7)$$

The exponent $k_d$ and the weights $w_n$ are set by the user, according to the importance that shall be given to the parameters $v_n$, that represent:

- $v_d$ : Pass duration
- $v_e$ : Max. elevation as seen by the receiver
- $v_\rho$ : Reciprocal of the min. distance from the receiving station
- $v_L$ : Reciprocal of the min. radar signal loss
- $v_m$ : Reciprocal of the min. optical magnitude

### *2.3.3. Genetic algorithm*

The schedule can be obtained as a solution of an optimization problem (maximize $\boldsymbol{s} \cdot \boldsymbol{f}$ ) subject to a constraint ($\mathbf{s}^{\mathbf{T}}\mathbf{M}\,\mathbf{s} = 0$).

The Boolean representation that has been introduced is particularly suitable for binary genetic algorithms.

SΞNSIT features a genetic strategy based upon the DEAP library [9]. Specifically, it starts by creating *popsize* schedules, instantiated as follows:

1. Start from an empty schedule.
2. Add all the passes that do not have conflicts (e.g., passes on survey sensors).
3. Add other passes one at a time, while checking that no conflicts occur.
4. Stop when no other pass can be added without introducing conflicts in the schedule.

Afterwards, for a given number of times (called *generations*), the following steps are executed:

1. **Selection**: select the best schedule among 3 randomly chosen schedules, *popsize* times (to keep the size of the population unaltered).
2. **Cross-over**: randomly choose two schedules and swap a segment of them, leading to the addition and removal of passes; the other conflicting passes are removed.
3. **Mutation**: randomly add or remove passes to observe in the schedules; when a pass is added, any conflicting passes are removed.
4. **Refill**: randomly add non-conflicting passes to a schedule, until possible (similarly to steps 3 and 4 of the procedure that creates the original schedules).

Since all the steps are guaranteed to generate conflict-free schedules, it is not necessary to introduce other means of enforcing the constraint $\mathbf{s}^{\mathbf{T}}\mathbf{M}\,\mathbf{s} = 0$ (such as penalties in the *fitness*).

The software allows to tune several parameters of the optimization through the GUI (*Fig. 13*), such as:

- The weights associated to the fitness values.
- The size of the population.

- The number of generations.
- The probabilities of the random processes (cross-over, mutation, refill).
- The initial seed for the random number generator (in order to achieve repeatable results).

The genetic scheduler tab shows the progress of the algorithm in real-time: it reports statistics and plots about the scores of the schedules that are being produced.

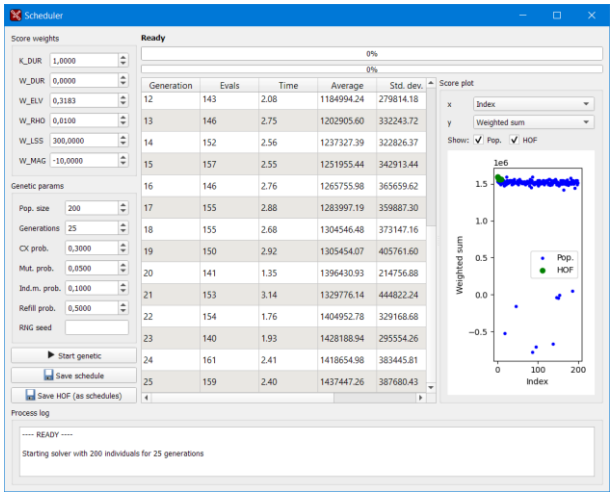


*Fig. 13 SΞNSIT GUI: genetic scheduler tab.*

Throughout the process, a list of the best 10 schedules, the *hall of fame*, is kept in memory. The user may explore the schedules, either as a list of passes (*Fig. 14*) or as a timeline (*Fig. 15*). The latter shows the chosen parts of passes (in red) and the not-chosen ones (in white). The overall timeline of scheduled passes for each sensor is represented in blue at the top of the graph.

It is also possible to export the obtained schedule in CSV format, ready to be executed by sensors.

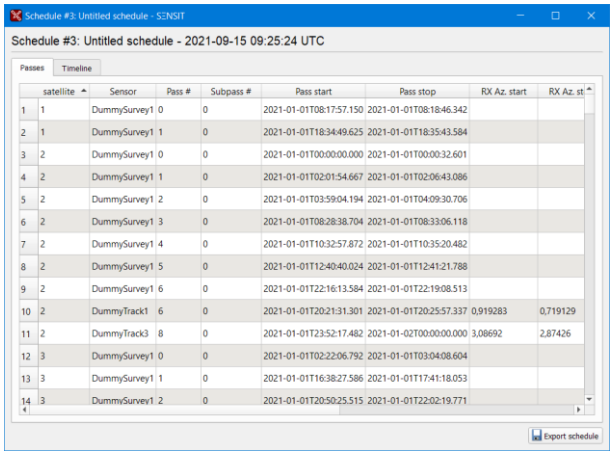


*Fig. 14 SΞNSIT GUI: list of scheduled passes.*

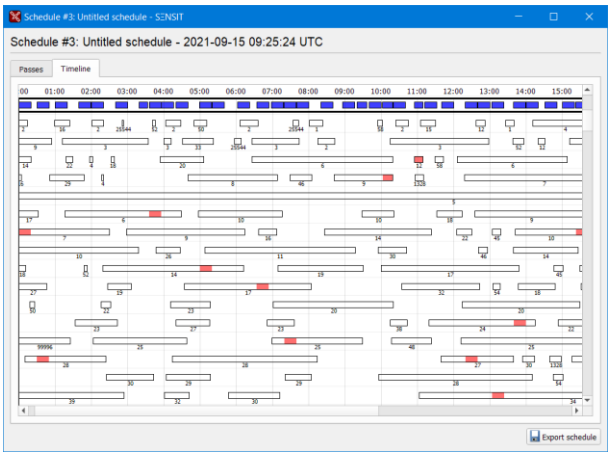


*Fig. 15 SΞNSIT GUI: timeline showing scheduled portions of passes (in red).*

### *2.4. Catalogue build-up*

The third module processes the scheduled passes in order to build-up and maintain the network catalogue (*Fig. 16*).

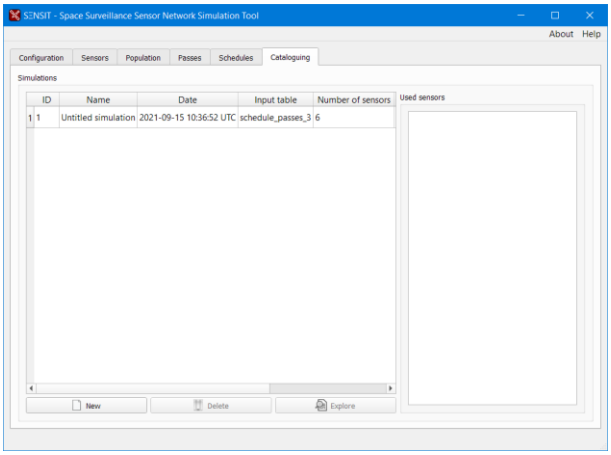


*Fig. 16 SΞNSIT GUI: cataloguing tab.*

It is necessary to select one of the schedules produced previously, that will be used for the cataloguing simulation. The user can decide to consider a subset of sensors: in this way, it is possible to conveniently execute different simulations and compare the results.

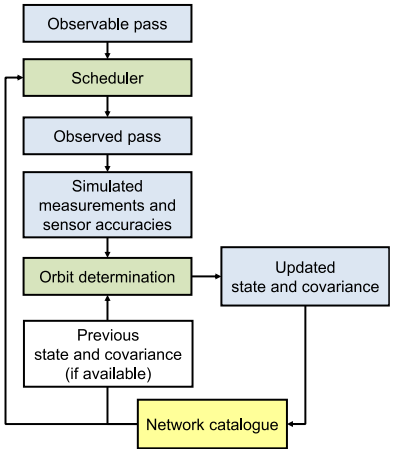


*Fig. 17 Schematic representation of the catalogue build-up and update process.*

The catalogue build-up process (*Fig. 17*) features a basic scheduler that checks again the rules already enforced by the genetic scheduler. It is therefore possible to execute this process not only from a conflict-free schedule, but also from the overall list of observable passes. This can be used to simulate a network that instead of scheduling the observations in advance, decides what pass to observe one by one.

This basic scheduler checks an additional rule: sensors in tracking mode can observe only the passes of objects already in the catalogue and with a covariance compatible with the user defined thresholds. This is enforced to simulate the fact that, in order to correctly point towards the object, it must be previously known with a certain accuracy.

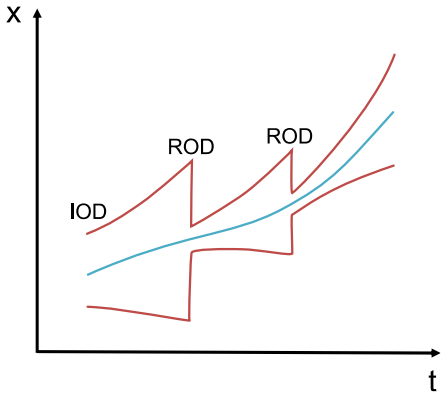


*Fig. 18 Schematic representation of the IOD and ROD processes, blue line = mean state, red line = uncertainty.*

At the beginning of the simulation, the network catalogue contains only the objects for which the user provided an initial state covariance.

When a non-catalogued object is observable by a sensor in survey mode, Initial Orbit Determination (IOD) is performed (*Fig. 18*). Provided that the diagonal components of the state covariance in the QSW reference frame are below a user-defined threshold, the object will join the network catalogue.

The condition on the covariance is checked throughout the simulation since position and velocity uncertainties enlarge over time.

When a catalogued object is observable, the basic scheduler decides whether the pass is actually observed. If this is the case, Refined Orbit Determination is executed, that updates the covariance matrix of the object.

In order to simplify the process, the mean state of the object obtained by the orbit determination process is not recorded: instead, it is always evaluated by SGP4 propagation of the elements provided by the user, adding noise when necessary.

Concerning the OD sub-module two different pipelines have been proposed: the Estimated covariance OD and the Non-Linear Least Squares OD. The former ensures a reduced computational cost by estimating only the expected outcome of an OD process in terms of covariance. The latter instead performs the whole orbit determination process using the Non-Linear Least Squares optimization algorithm.

### *2.4.1. Catalogue build-up and maintenance process*

Before moving on, it is worthwhile to explain how the covariance is assessed to be valid or not.

Firstly, the rotation matrix $\mathbf{R_{E2Q}}$ from the Earth-Centered Inertial (ECI) reference frame to the QSW frame [7] is computed. The QSW frame is composed of the unit vectors defined as:

- $\vec{q}$ : collinear to the geocentric satellite position (from the planet center to the spacecraft)
- $\vec{w}$ : collinear to the orbital kinetic momentum (normal to the orbital plane)
- $\vec{s}$ : equal to $\vec{w} \wedge \vec{q}$

The covariance matrix $\mathbf{C_Q}$ in QSW frame is obtained from the one in ECI frame $\mathbf{C_E}$ as in *Eq. 8*.

$$\mathbf{C_Q} = \mathbf{R_{E2Q}}\, \mathbf{C_E}\, \mathbf{R_{E2Q}}^{\mathbf{T}} \qquad (8)$$

The object enters or remains in the network catalogue if the first three diagonal components of $\mathbf{C_Q}$ are lower than a specified threshold, that must be defined by the user and may be different according to the altitude of the object.

Next, if the object has already a valid covariance $\mathbf{C_{E,t_{i-1}}}$ at the previous $\boldsymbol{t_{i-1}}$ ROD or IOD instant, it is propagated up to the initial observation epoch and $\mathbf{C_{E,t_i}}$ is obtained, as shown in *Eq. 9*.

$$\mathbf{C_{E,t_i}} = \mathbf{J_{kep}}\, \mathbf{C_{E,t_{i-1}}}\, \mathbf{J_{kep}}^{\mathbf{T}} \qquad (9)$$

$\mathbf{J_{kep}}$ is the Jacobian matrix referred to the Keplerian propagation from $\boldsymbol{t_{i-1}}$ to $\boldsymbol{t_i}$. If $\mathbf{C_{E,t_i}}$ is still within the thresholds, ROD will be performed.

For this stage, as stated in *Sec. 2.4*, two approaches have been implemented.

The first one is the Estimated Covariance OD, that requires:

- $\mathbf{C_{E,t_i}}$: the known state covariance matrix,
- $\mathbf{C_m}$: the covariance matrix of the measurements (sensor dependent),
- $\mathbf{J_{m/s}}$: the Jacobian matrix of the measurements with respect to the propagated states of the object,
- $\mathbf{J_{s/s_0}}$: the Jacobian matrix of the propagated states with respect to the initial state, approximated as the state transition matrix of a Keplerian propagation.

The updated covariance $\mathbf{C_{E,t_{i,new}}}$ is computed as reported in *Eq. 10*.

$$\mathbf{C_{E,t_{i,new}}} = \left(\mathbf{J_{m/s_0}}^{\mathbf{T}}\,\mathbf{C_m}^{-1}\,\mathbf{J_{m/s_0}} + \mathbf{C_{E,t_i}}^{-1}\right)^{-1} \qquad (10)$$

The Jacobian matrix of the measurements with respect to the initial state $\mathbf{J_{m/s_0}}$ is computed as in *Eq. 11*.

$$\mathbf{J_{m/s_0}} = \mathbf{J_{m/s}}\,\mathbf{J_{s/s_0}} \qquad (11)$$

The other OD formulation relies on a Non-Linear Least Squares optimization [10]; the algorithm has been designed as follows:

- Synthetic measures $\mathbf{m_{ref}}$ are generated with a fixed time step within the observation window, and Gaussian noise is added according to the sensor accuracy.
- The initial state guess $\mathbf{s_0}$ is computed by SPG4 propagation of the elements of the object and by the addition of Gaussian noise, according to the state covariance $\mathbf{C_{E,t_i}}$.
- An iterative procedure propagates the initial state $\mathbf{s_0}$ with Keplerian dynamics up to the time instants of the measurements, and projects it to the measurement space $\mathbf{m_{LS}}$. A design matrix $\mathbf{D}$ is built using the Jacobian matrix of the measurements with respect to the initial state $\mathbf{J_{m/s_0}}$ and weights $\mathbf{W}$ (defined from the sensor accuracy) (*Eq. 12*). The initial state with its covariance is considered as *a priori information*.

$$\mathbf{D} = \mathbf{W}\,\mathbf{J_{m/s_0}} \qquad (12)$$

The cost function is the measures residual $\mathbf{r}$ (*Eq. 13*).

$$\mathbf{r} = \mathbf{m_{ref}} - \mathbf{m_{LS}} \qquad (13)$$

The vector $\mathbf{r}$ and the matrix $\mathbf{D}$ are intended to solve the normal equation, that outputs $\mathbf{C_{E,t_{i,new}}}$ and the $\mathbf{s_0}$ correction factor.

The routines stops if a maximum number of iterations or convergence is reached.

If the object is out of catalogue and passes over a survey ground station, IOD will be conducted in similar manner as illustrated for ROD.

The differences are:

- For the Estimated Covariance OD, the updated covariance is determined as in *Eq. 14*.

$$\mathbf{C_{E,t_{i,new}}} = \left(\mathbf{J_{m/s_0}}^{\mathbf{T}}\,\mathbf{C_m}^{-1}\,\mathbf{J_{m/s_0}}\right)^{-1} \qquad (14)$$

- For the Non-Linear Least Squares algorithm, no *a priori information* is considered.

After ROD or IOD, the updated covariances and the corresponding epochs are saved in the program database.

## 3. Results

The results from the previous steps are stored in the application database file. These are employed to illustrate data as different interactive graphs, designed to maximize the user awareness of the network performance. Since it is encoded in the well-documented SQLite format, the user is free to analyze its content using other software.

### *3.1. Sensor network performance analysis*

The representations are both sensor-oriented and population-oriented, to have an organic view of the results of the simulation. The available visualizations are described hereafter.

#### *3.1.1. Pass list*

The first view that is shown to the user is a list of the observed passes, with the ID of the object, the epoch, the sensor name, the orbit determination type and the resulting covariance. This allows to analyze in detail the observations performed by each sensor.

#### *3.1.2. Redundancy matrix*

The redundancy matrix (*Fig. 19*) is a table that shows the ratio of objects visible from a given sensor (the one on the row) that can also be seen by another (the one on the column). This can help in determining the redundancy of the sensors.

| | DUMMY1 | DUMMY2 | DUMMY3 | DUMMY4 |
|---|---|---|---|---|
| DUMMY1 | 100% | 97.8% | 100% | 62.4% |
| DUMMY2 | 100% | 100% | 100% | 62.6% |
| DUMMY3 | 100% | 97.8% | 100% | 62.4% |
| DUMMY4 | 100% | 98.3% | 100% | 100% |

*Fig. 19 Redundancy matrix.*

### *3.1.3. Catalogue population plot*

An example of population-focused plot is displayed in *Fig. 20*. It shows a Cartesian plane, having two orbital parameters as axes. X and Y can be modified by the user and the distribution changes accordingly, allowing to understand the orbital regime they belong to. The color of each point varies according to the number of times it was observable by the sensors.

It is possible to select only a subset of sensors, and the color intensity of the points changes accordingly: this lets the user understand the importance of each sensor in the observations.

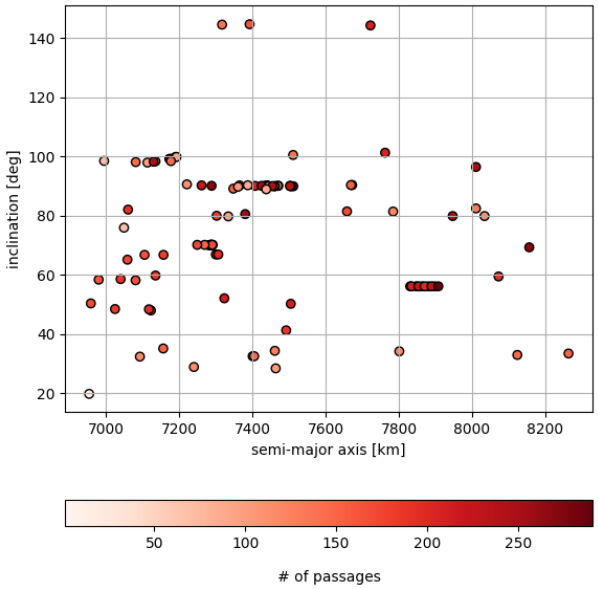


*Fig. 20 Catalogue population plot.*

### *3.1.4. Catalogue evolution*

A further point of view about the interaction between the simulated population and the network is the catalogue evolution representation (*Fig.* 21), in terms of percentage of objects belonging to the reference population.

The entries are determined by a successful initial orbit determination, while the exits by the covariance exceeding the thresholds.

The plot portrays two different population trends: the blue one refers to the total amount of objects belonging to the catalogue, while the orange one depicts the evolution of a subset of objects located in a specific orbital regime selected by the user.

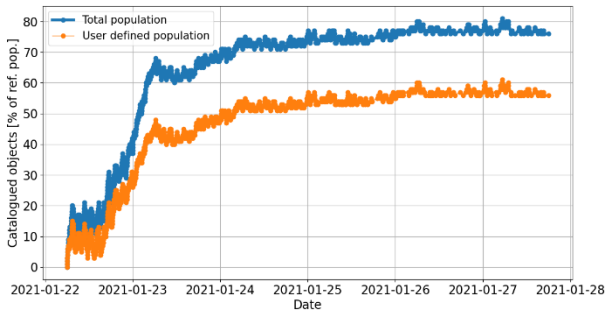


*Fig.* 21 *Evolution of catalogued population over time.*

### *3.1.5. Covariance evolution*

Knowing the covariance evolution in time of a specific target is crucial to understand if it is inside or outside the sensor network catalogue within a simulation.

In *Fig. 22* a Cartesian plot describes the trend of a catalogued object covariance over time, using the square root of its trace as metric. The orange dots represent the IODs, while the green ones the RODs.

The user can also choose to see separately the first three diagonal components of the covariance matrix in the QSW reference frame.

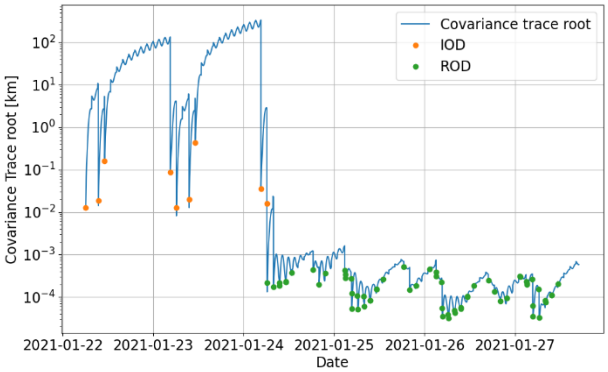


*Fig. 22 Covariance trend over time.*

### *3.1.6. Coverage pie chart*

A clear view of the contributions of a subset of sensors to the entire network performance can be pictured as a pie chart (*Fig. 23*): each slice represents the percentage of simulated objects seen only by the corresponding sensor or combination of sensors (without any intersection among them, due to the rule used to

determine the contributions). The user can select up to three sensors at the same time.

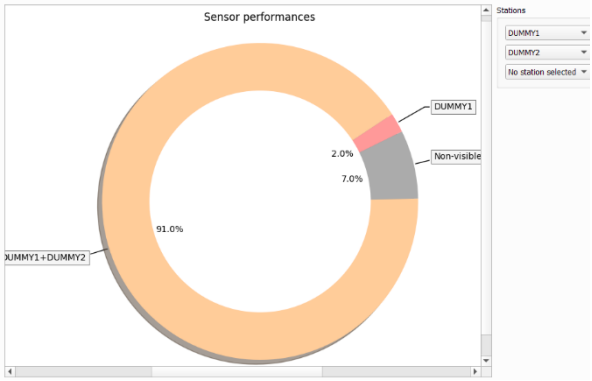


*Fig. 23 Pie chart plot.*

### *3.1.7. FOV projection*

In order to have an organic view of the network coverage over areas of interest, the plot in *Fig. 24* provides a geographical projection of the FOV of survey sensors at different altitudes. The representation depends on the sensor position, the FOV size and shape (rectangular or elliptical), the sensor pointing and the intersection altitude. Coverage areas are colored according to their type (optical or radar) and their size changes according to the intersection altitude, that can be modified by the user in real-time. This plot can also be useful to understand overlaps between the FOVs of survey sensors.

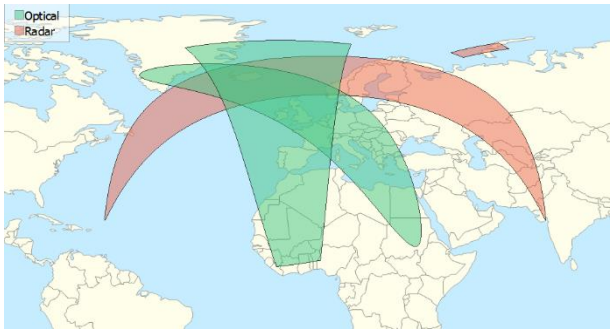


*Fig. 24 Coverage areas.*

### *3.1.8. Maximum re-observation time*

It is important to have a sense of what is the maximum re-observation time (i.e., the time elapsing between two consecutive observable passes) for all the objects passing over a ground station. This is instrumental to figure out how a sensor can contribute to the catalogue maintenance, since the lower the re-observation times the more the network can keep up to date the covariance estimates in the catalogue.

The graph in *Fig. 25* illustrates the cumulative distribution of the maximum re-observation times of all the objects transiting over a user defined sensor or over the entire sensor network.

On top of that, a vertical line set at 24 hours splits the distribution in two parts, to highlight the percentage of objects which can always be seen within one day or less.

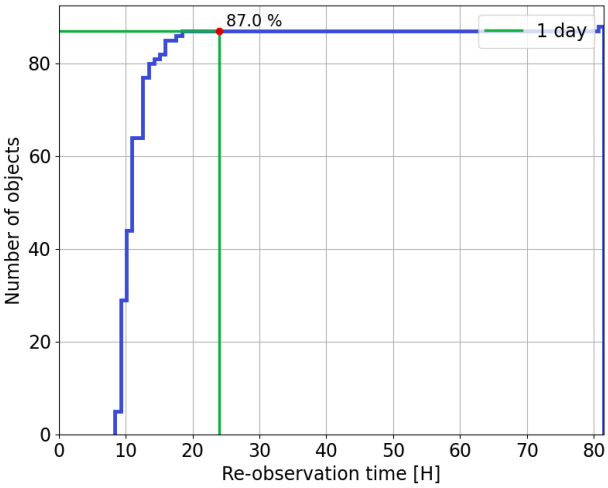


*Fig. 25 Maximum re-observation times.*

### *3.1.9. Coverage histogram*

The coverage histogram (*Fig. 26*) permits to evaluate the coverage of specific sensors with respect to different orbital parameters. The user can choose an orbital parameter (for the horizontal axis) and up to three sensors to compare their coverage. Each sensor is described as a bar distribution, whose height represent the percentage of observable objects within that range of the selected orbital parameter.

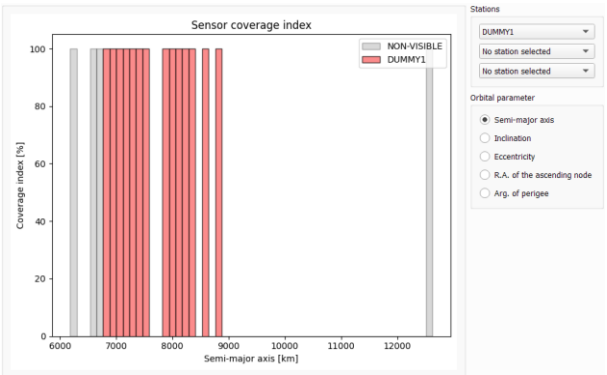


*Fig. 26 Coverage histogram.*

## 3.2. *Computational time*

The goal of SΞNSIT is not merely to provide an accurate sensor network modeling, but also to output the expected results in a reasonable time frame.

The analysis of the computational time has been conducted on a PC featuring a 3700x AMD processor, with 8 physical and 16 logical cores, and 16 GB of RAM.

Several sensor network simulations have been performed by setting the following parameters:

- Number of input objects: 10, 100, 1000.
- Number of involved sensors: 1, 5, 10
- Simulation time frame: 3 days

*Table 1* and *Fig. 27* outline the time required for the computation of observable passes (*Sec. 2.2*).

Time clearly scales up when increasing the number of objects, while it is less affected by the amount of sensors thanks to the implemented optimizations (mainly, the cache of object states).

*Table 1 Time required by pass computation, in seconds.*

| | 1 sensor | 5 sensors | 10 sensors |
|---|---|---|---|
| **10 obj.** | 3 | 4 | 4 |
| **100 obj.** | 21 | 23 | 25 |
| **1000 obj.** | 204 | 223 | 244 |

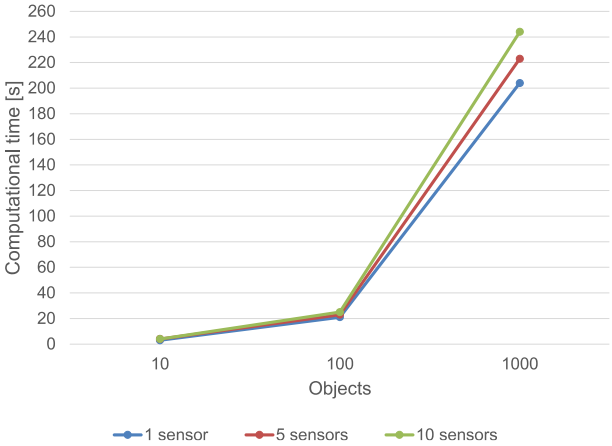


*Fig. 27 Time required by computation of observable passes.*

*Table 2* and *Fig. 28* report the time needed to perform catalog build-up and maintenance using Estimated Covariance OD, starting from the whole list of passes (not from a conflict-free schedule). These times are influenced by both objects and sensors.

*Table 2 Time required by catalogue build-up using estimated OD, in seconds.*

| | 1 sensor | 5 sensors | 10 sensors |
|---|---|---|---|
| **10 obj.** | 1 | 1 | 1 |
| **100 obj.** | 3 | 3 | 3 |
| **1000 obj.** | 19 | 30 | 83 |

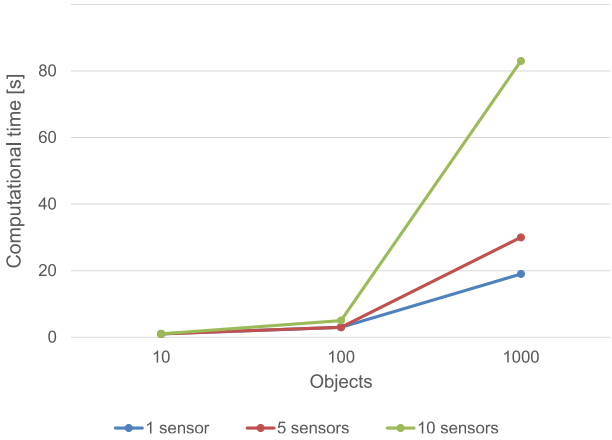


*Fig. 28 Time required by catalogue build-up using Estimated OD.*

*Table 3* and *Fig. 29*, instead, report the time needed to perform catalog build-up and maintenance if Non-Linear Least Squares OD is used, starting from the whole list of passes also in this case.

The reported times are approximately double with respect to the ones required by Estimated Covariance OD.

*Table 3 Time required by catalogue build-up using NLS OD, in seconds.*

| | 1 sensor | 5 sensors | 10 sensors |
|---|---|---|---|
| **10 obj.** | 1 | 1 | 1 |
| **100 obj.** | 4 | 6 | 12 |
| **1000 obj.** | 40 | 63 | 191 |

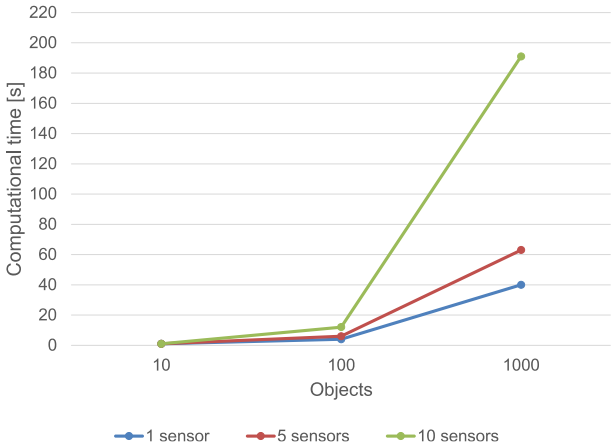


*Fig. 29 Time required by catalogue build-up using NLS OD.*

The genetic scheduler has proven to produce good schedules in few minutes, yet the precise execution time is not reported here since it is affected by many parameters (e.g., number of passes, population size, mutation and cross-over probabilities).

In the considered conditions, the overall time required by SΞNSIT to compute passes, generate a schedule and

simulate cataloguing is in the order of minutes, therefore it appears suitable for fast simulations.

## 4. Conclusions and future work

SΞNSIT, the program developed in this work, answers the need of having a software tool that allows to model SST sensor networks and evaluate their performance, in terms of coverage and capability of building and maintaining a catalogue of space objects. It also features an optimal scheduler, that may be used to schedule the observations of existing sensors.

Moreover, SΞNSIT allows to perform these tasks in a user-friendly way, thanks to its Graphical User Interface, the capability of running through different operative systems and the speed of execution.

Yet, several improvements can still be implemented in SΞNSIT. The recent introduction of the genetic optimization algorithm requires for further research, in order to determine the most suitable parameters (e.g., weights, genetic strategy). Afterwards, the developers are considering introducing a batch process for the automatic generation of new sensors according to given criteria (e.g., different locations for a sensor within a given area, or different type of sensors in a fixed position).

An improved simulation of observations is also being studied, with the ability to consider customizable radar beam patterns, tracking errors due to the inaccuracy of the ephemerides, random events that could prevent observability (e.g., adverse weather conditions, maintenance downtime).

**Acknowledgements**

The results of this project have been supported by the agreement of the Italian Space Agency and the National Institute for Astrophysics on Space Debris (Detriti Spaziali – Supporto alle attività IADC e SST 2019-2021, n. 2020-6-HH.0).

**References**

[1] "Space debris - evolution in pictures," [Online]. Available: https://www.esa.int/About_Us/ESOC/Space_debris_-_evolution_in_pictures.

[2] ESA, "Debris object evolution," [Online]. Available: https://www.esa.int/ESA_Multimedia/Images/2017/04/Debris_object_evolution.

[3] V. Morand, C. Yanez and J. C. Dolado Perez, "BAS3E: A framework to Conceive, Design, and Validate Present and Future SST Architectures," in *First International Orbital Debris Conference*, 2019.

[4] N. O. Gómez, I. A. Gómez and S. A. Vildarraz, "Architectural description of the Spanish Space Surveillance and Tracking System," 2017.

[5] G. Purpura, A. De Vittori, R. Cipollone, M. Massari, C. Colombo, P. Di Lizia, S. Cicalò, F. Guerra, A. Bertolucci, A. Di Cecco and L. Salotti, "Development of a Software Suite for Performance Assessment of SST Sensor Networks," in *8th European Conference on Space Debris*, 2021.

[6] C. Acton, "Ancillary Data Services of NASA's Navigation and Ancillary Information Facility," *Planetary and Space Science*, vol. 44, no. 1, pp. 65-70, 1996.

[7] CCSDS Secretariat, National Aeronautics and Space Administration, *Navigation Data - Definitions and Conventions,* Washington, DC, 2019.

[8] D. W. E. Green, "Magnitude corrections for athmospheric extinction," July 1992. [Online]. Available: http://www.icq.eps.harvard.edu/ICQExtinct.html.

[9] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau and C. Gagné, "DEAP: Evolutionary Algorithms Made Easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171-2175, jul 2012.

[10] A. Milani and G. F. Gronchi, Theory of Orbit Determination, Cambridge University Press, 2010.