COMPUTATIONAL TOOL FOR SATELLITE SITUATIONAL ANALYSIS: CURRENT STATUS AND FUTURE HORIZONS

Atanas Atanassov

Space Research and Technology Institute – Bulgarian Academy of Sciences
e-mail: At_M_Atanassov@yahoo.com

***Keywords**: satellite situational analysis, orbital events, time intervals determination*

***Abstract**: Satellite situational analysis involves determining the specific time intervals during which orbital conditions are satisfied to perform spacecraft operations, such as scientific experiments or data acquisition. To achieve this, the analytical framework is formulated as a Constraint Satisfaction Problem (CSP), typically encompassing multiple geometric and physical constraints. Conducting situational analysis is essential across various stages of space mission design, planning, and execution. Within this scope, two distinct yet interconnected software modules have been implemented: a program editor for task composition and a parallel situational solver for task execution, the core design principles of which are outlined in this paper. In multi-satellite constellations equipped with diverse scientific instruments, repeated situational analysis is required – particularly during the early phase of mission architecture definition – demanding substantial computational resources. To address this challenge, the developed software tool is parallelised and incorporates dynamic heuristic approaches to enhance computational efficiency. Finally, the performance of the situational solver is illustrated through the resolution of specific situational tasks governing the management of a satellite power subsystem and multiple payloads.*

ИЗЧИСЛИТЕЛЕН ИНСТРУМЕНТ ЗА СПЪТНИКОВ СИТУАЦИОНЕН АНАЛИЗ: ТЕКУЩО СЪСТОЯНИЕ И ПЕРСПЕКТИВИ

Атанас Атанасов

Институт за космически изследвания и технологии – Българска академия на науките
e-mail: At_M_Atanassov@yahoo.com

***Ключови думи:** спътников ситуационен анализ, орбитални събития, определяне на времеви интервали*

***Резюме:** Спътниковият ситуационен анализ се занимава с определяне на времеви интервали, които са подходящи за провеждане на операции/активности свързани с добив на данни/информация необходими за решаване на конкретна научна или технологична задача. За целта се решават задачи от типа „задача изпълнение на ограничения“. Броят на ограниченията е едно или няколко. Извършването на ситуационен анализ е от съществено значение на различни етапи от подготовката и реализацията на космическите мисии.*

За извършване на спътников ситуационен анализ са разработени два отделни, но взаимно свързани инструмента – единият, редактор за създаване на модели на ситуационните задачи и ситуационен процесор за тяхното решаване. В доклада са разгледани основните принципи за изграждане на двата инструмента. При космически мисии с голям брой спътници, научни инструменти и научни задачи, се налага многократно извършване/провеждане на ситуационен анализ, особено на началните етапи от подготовката на космическите мисии. Това налага изразходването на значително количество изчислителни ресурси. За тази цел, разработеният програмен инструмент е паралелизиран и включва евристични подходи за увеличаване на изчислителната ефективност.

Работата на ситуационния процесор е илюстрирана чрез решаване на ситуационни задачи, имащи отношение към управлението на енергийна спътникова подсистема и няколко инструмента.

Introduction

Computer simulations are utilised to clarify various aspects related to satellite orbital motion, the design and functional parameters of space and ground-based subsystems, and payloads at different stages of space mission preparation. This analysis falls under a distinct scientific field within space research known as Space Mission Analysis and Design (**SMAD**) [1]. Situational analysis (**SA**) is a crucial component of **SMAD**, determining time intervals or specific moments when conditions are suitable for conducting satellite operations or other orbital events.

The Parallel Situational Analysis Solver is currently under development at the Space Research and Technology Institute of the Bulgarian Academy of Sciences (BAS) [2]. Individual stages in the solver's development are presented in [2-8]. This paper presents an application of the situational solver in simulating the operation of several instruments.

Problem formulation

Situational analysis investigates and establishes whether the conditions required to perform satellite operations (such as measurements, experiments, communications, or space debris close-approach risks) are met during orbital motion. A single situational task ST may comprise one or more conditions sc:

(1) $$ST = sc_1 \wedge sc_2 \wedge \ldots \wedge sc_n = \bigwedge_1^n sc_i,$$

The conditions sc_i are predicate functions that evaluate to either true or false. Generally, they can be represented as:

$$sc_i = sc_i\left(\{\vec{R}\}, \{\alpha\}, \{\beta\}, t\right)$$

where $\{\vec{R}\}$ is the set of radius vectors of the objects, $\{\alpha\}$ is the set of parameters of the mathematical model describing the situational condition, and $\{\beta\}$ is the set of specific constraints. Within this formulation, solving a situational problem involves verifying the feasibility of logical functions at appropriate (or equidistant) instants of time t.

Program Editor for Situational Problem Composition

This editor is used to define, formulate and compose situational tasks. This process is based on previously developed and programmatically implemented models of situational conditions. To implement this editor, a **dialogue box** incorporating various controls is used for selecting and entering the parameters of the situational conditions. Depending on the type of space mission, a **list control** provides a corresponding set of situational conditions. Based on the selected conditions, the respective parameters $\{\alpha\}$ and constraints $\{\beta\}$ are displayed on the **dialogue box** by **static text controls**, allowing their values to be entered into **edit controls**. Input validation is applied to the entered values based on the domains defined for each parameter and constraint during the development of the respective situational condition. This prevents the entry of incorrect values and mitigates runtime program errors.

Auxiliary tools (implemented via **push-buttons**) have been developed to modify the sequence of situational conditions within a situational task or to remove a previously entered condition. Once the respective parameter values and constraints are configured, individual conditions, tasks, or sets of tasks can be saved for future use. When defining situational tasks for multi-satellite missions, a task can be assigned to either an individual satellite or the entire constellation. This capability is essential for heterogeneous missions, where distinct satellites perform specific functions.

Once the definition of the situational tasks for a given mission is complete, several critical operations are performed:

- The number of threads required to instantiate the parallel situational solver is entered into edit controls, and the solver itself is instantiated before exiting the dialog. A dedicated subroutine generates these threads, alongside two event objects per thread, which are utilized to synchronise the threads with the parent process. The handles for both the threads and the synchronisation events are stored within a two-dimensional array representing the solver's configuration parameters. Additionally, a global synchronisation event is created to manage inter-thread coordination when fetching the next situational task or group of tasks for processing. The number of situational tasks fetched concurrently is determined by the "granularity" parameter, the value of which is configured alongside the thread count.
- Prior to exiting the main subroutine of the situational editor, all solver control parameters (such as the thread count and solver configurations) along with the situational task data are passed via a global data area (common area) to a wrapper subroutine, which serves as the entry point

for the solver's threads. By utilizing different wrapper subroutines – the names of which are passed to the thread-creation subroutine and are essential for instantiating the threads themselves - various configurations of the situational solver can be implemented.

- All solver parameters and memory addresses containing the situational tasks, along with the satellite coordinate addresses (and other data), are recorded into a specific derived data type structure. This structure is subsequently passed to a subroutine possessing polymorphic properties. This universal subroutine registers solver descriptors into specific groups or "layers", reflecting the comprehensive architecture of space mission simulation models as a "System of Systems". The resulting descriptor contains all the necessary situational task data concerning a specific space mission. It is utilized both by the editors of other solvers to build interfaces between different interdisciplinary models within a multiphysics software environment, and by the simulation model interpreter for a given space mission.

Parallel Situational Solver

The development of the situational solver originated from a serial codebase, which was subsequently parallelised using the thread pool programming model [9]. This specific programming model was selected due to the following factors:

- **Irregular workloads:** The mathematical models governing various situational conditions are inherently non-uniform, leading to irregular computations. In certain scenarios, these irregularities arise within the situational conditions themselves at specific segments of the satellite trajectory; for instance, verifying whether a satellite is in a sunlit region, the Earth's shadow (umbra), or penumbra requires varying numbers of processor operations.
- **Memory latency:** The spatial allocation of data required to solve situational tasks within the address space (such as satellite coordinates and task parameters) can lead to non-uniform memory latencies during data retrieval, causing processor stalls. These processes are inherently dependent on the underlying processor architecture and the operating system.
- **Load balancing:** To prevent system overload and mitigate resource constraints, it is recommended that the number of threads within the pool does not exceed the total number of available CPU cores. This restriction is crucial, as an additional master thread is dedicated to managing the simulation loop.

Consequent to these challenges, the thread pool model provides dynamic scheduling of computations to achieve maximum efficiency and optimal hardware utilisation. In practice, this model was further developed by incorporating synchronisation mechanisms with the parent thread (which controls the overall simulation loop) and mitigating potential race conditions. Furthermore, the solver management logic utilizes the "granularity" parameter to dispatch a predefined number of situational tasks to available worker threads. When the remaining unsolved tasks fall below this threshold, the available threads dynamically shift to fetching tasks individually.

To coordinate the thread pool operation, a universal subroutine is invoked by the solver's wrapper subroutines. This subroutine exhibits polymorphic properties, enabling it to be invoked with varying actual parameters tailored to the specifications of the different solvers that utilize it. The entry point of the main solver subroutine is encapsulated within the wrapper subroutine; it is passed to the thread pool management subroutine, which subsequently executes the core algorithm that processes the situational tasks. From an algorithmic perspective, this core subroutine operates as a finite-state machine that accepts an ordered set of situational conditions as an input tape for a given situational task. Each situational condition possesses a unique identifier, which dictates the invocation of the corresponding subroutines responsible for evaluating those specific conditions.

Examples of Implemented Situational Conditions

At this stage of development, several models for specific situational conditions have been formulated and programmatically implemented within the parallel solver framework. Algorithms have been developed to precisely determine the boundaries (start and end points) of the target time intervals during which these situational conditions are satisfied. To achieve this high precision, the solver retains the state vectors from the last few preceding time steps, alongside the satellite coordinates calculated at the end of the current integration step. These state vectors serve as interpolation nodes. Consequently, interpolation within the bounds of the final orbital integration step is utilized to accurately pinpoint the entry and exit instants of the intervals.

The implemented situational conditions include algorithms to determine the time intervals during which a satellite is positioned:

- **Subject to spatio-temporal conditions** [3]: These apply to a pair of satellites traversing the same spatial region with a time variance Δt that is strictly less than a characteristic timeframe τ associated with the physical process under investigation.
- **Outside the moonlit segment** of the orbit [4].
- **Within the Earth's shadow (umbra) or the sunlit region** [5]: Both scenarios utilize a conical model of the Earth's shadow, with an additional verification layer to detect transitions through the penumbra.
- **Within a high-risk close-approach zone** [6]: These conditions are developed to detect potential conjunctions and close approaches between active satellites, as well as between satellites and space debris.
- **Within the line-of-sight (radio visibility) zone** of a designated ground station [7].
- **In close angular proximity to the Sun:** This addresses Sun outage scenarios for a communicating satellite pair, verifying whether the transmitting satellite falls within a narrow angular separation from the Sun as viewed by the receiver.
- **Above a circular segment of the Earth's surface**, defined by specified coordinates for its centre $((\varphi, \lambda))$ and an angular radius Θ.

Efficiency Enhancement via Dynamic Heuristic Reordering

According to Equation (1), the primary objective is to determine the time intervals during which all situational conditions are simultaneously satisfied. Conversely, the complementary problem $\overline{SP}$ can be formulated to identify the time intervals where at least one condition is violated:

(2) $$\overline{ST} = \overline{sc(t)_1} \vee \overline{sc(t)_2} \vee \ldots \vee \overline{sc(t)_k} \vee \ldots \vee \overline{sc(t)_n}$$

To satisfy this complementary formulation, it is sufficient for just a single condition $\overline{sc(t)_k}$ to evaluate to $\boldsymbol{true}$ (meaning the original condition $sc(t)_k$ is violated). Once a specific k-th condition is identified as sufficient to satisfy the negative task state $\overline{sc(t)_k} = true$, the evaluation sequence for subsequent time steps can be optimized by applying the following reordering heuristic [8]:

(3) $$\overline{ST} = \overline{sc(t+\triangle t)_k} \vee \overline{sc(t+\triangle t)_1} \vee \ldots \vee \overline{sc(t+\triangle t)_{k-1}} \vee \overline{sc(t+\triangle t)_{k+1}} \vee \ldots \vee \overline{sc(t+\triangle t)_n}$$

The underlying rationale for this reordering stems from the fact that situational conditions are slowly varying functions relative to the simulation model's temporal discretization step Δt. Consequently, shifting the previously unfulfilled $\boldsymbol{k}^{-th}$ condition to the first position in the evaluation queue allows the solver to trigger a short-circuit

Precise Determination of Time Interval Boundaries

In space mission simulations, the numerical integration of satellite equations of motion is executed using a discrete temporal step Δt. To ensure computational efficiency, this step typically ranges from tens of seconds to a minute or more, during which low-Earth orbiting satellites significantly alter their spatial positions. During situational analysis, the parameters $\alpha(t)$ governing the situational task conditions are evaluated. If the integration steps are substantial, the variations in these parameter values can also be significant. Consequently, evaluating the situational conditions solely at the discrete integration steps where the state vectors are solved, $SP(\alpha(t))$, can introduce considerable boundary errors when defining the start and end boundaries of the target intervals Δt^*. The actual entry and exit instants of these intervals reside somewhere within the bounds of the current integration step. Crucially, when $\Delta t^* < \Delta t$, an entire interval may be omitted entirely. This risk is particularly pronounced in high-precision tasks such as detecting equatorial or terminator crossings (the boundary dividing the sunlit and shadowed regions of the Earth) or analysing close-approach conjunctions between satellites and space debris.

To precisely isolate these intervals Δt^*, two distinct interpolation-based approaches have been developed, utilizing Lagrange interpolation [10] to approximate satellite coordinates and situational parameters at arbitrary instants within the integration step Δt. In the first method, the interval onset t^*_{start} is detected when the evaluated state of the situational task changes from $\boldsymbol{false}$-to-$\boldsymbol{true}$ across two consecutive steps, $ST(\alpha(t))$ and $ST(\alpha(t + \Delta t))$. Without refinement, the boundary defaults to $t^*_{start} = t + \Delta t$. Analogously, the interval termination t^*_{final} is identified via a $\boldsymbol{true}$-to-$\boldsymbol{false}$ transition, defaulting to $t^*_{final} = t$. The boundary refinement is executed iteratively by selecting successive sub-steps t_j within the window $(t, t + \Delta t)$ and evaluating $ST\left(\alpha(t_j)\right)$. This convergent iterative loop

terminates when $|t_{j+1} - t_j| < tol$, where tol defines the user-specified tolerance for boundary precision. The inherent limitation of this baseline approach remains its susceptibility to missing short-duration intervals where $\Delta t^* < \Delta t$.

The second method addresses this limitation by actively monitoring the mathematical trends of the parameters $\alpha(t)$ relative to their respective operational constraints β. Numerical differentiation is applied to evaluate the first derivatives of $\alpha(t)$ to detect inflection points or local extrema. A sign change in the derivative (e.g., from negative to positive) serves as a predictive indicator that triggers a deep search for a hidden interval Δt^*, even when $\Delta t^* < \Delta t$. Once triggered, root-finding algorithms are deployed within the macro step Δt, combining standard bisection techniques with specialized refinement heuristics. This iterative refinement process continues until convergence is reached. This advanced approach has been successfully implemented to resolve high-risk satellite close approaches (conjunctions) [7] and short-duration passes through the line-of-sight radio visibility zones of ground tracking stations.

Application Example

To illustrate the operation of the situational solver, several situational tasks required to manage two scientific instruments are solved. These instruments are defined as two cameras: one operating in the visible spectrum to observe the Earth's surface within the sunlit segments of the orbit, and the other operating in the infrared spectrum within the Earth's shadow. Two circular segments on the Earth's surface, Θ_1 and Θ_2, are selected with geographical coordinates and angular radii configured as $(33°, 1°, 9°)$ and $(66°, 1°, 9°)$, respectively.

Within the computer simulation, the situational task ST_1 is solved using a single condition, $sc_{1,1}$, which verifies whether the satellite is positioned in the sunlit part of its trajectory. This step is necessary to simulate electrical power generation from the solar arrays. To simulate the operation of the visible-spectrum instrument when the satellite traverses either Θ_1 or Θ_2, two situational tasks are evaluated, each comprising two conditions: $ST_2 = sc_{2,1} \wedge sc_{2,2}$ and $ST_3 = sc_{3,1} \wedge sc_{3,2}$. The conditions $sc_{2,1}$ and $sc_{3,1}$ verify proximity to Θ_1 and Θ_2, respectively. The conditions $sc_{2,2}$ and $sc_{3,2}$ are identical and check if the satellite remains within the sunlit segment of the orbit.

The second instrument operates based on the results of two situational tasks, each containing three conditions: $ST_4 = sc_{4,1} \wedge sc_{4,2} \wedge sc_{4,3}$ and $ST_5 = sc_{5,1} \wedge sc_{5,2} \wedge sc_{5,3}$. The primary conditions, $sc_{4,1}$ and $sc_{5,1}$, correspond to the regions Θ_1 and Θ_2, while the secondary conditions, $sc_{4,2}$ and $sc_{5,2}$, verify if the satellite is within the Earth's shadow. The tertiary conditions, $sc_{4,3}$ and $sc_{5,3}$, check if the Moon is below the satellite's horizon to ensure the absence of lunar light pollution. Figure 1 illustrates the time-sequence diagrams representing the solutions to these described situational tasks.

For the purpose of this demonstration, the satellite's orbital parameters are defined as follows: semi-major axis $\boldsymbol{a} = 7{,}200{,}000$ m, eccentricity $\boldsymbol{e} = 0.001$, inclination $\boldsymbol{i} = 0.0°$, right ascension of the ascending node $\boldsymbol{\Omega} = 0.0°$, and argument of perigee $\boldsymbol{\omega} = 0.0°$.

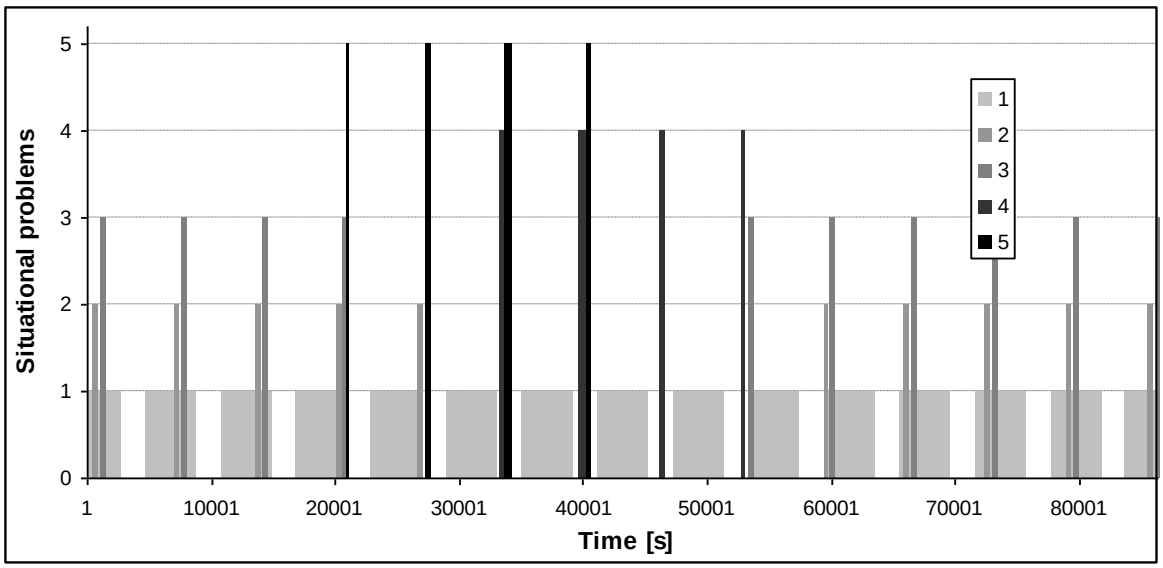


Fig. 1. Time-sequence diagrams representing the solutions to the five described situational tasks over a 24-hour period

To evaluate the efficiency of the dynamic heuristic reordering, two separate analyses of the configured situational tasks were conducted: one utilizing a static condition sequence and another applying the dynamic heuristic reordering strategy. The total number of evaluated situational conditions was **65,153** for the static approach and **44,342** for the dynamically reordered execution. It should be noted that in the static baseline scenario, the total count of evaluated conditions heavily depends on their initial arrangement and the relative position of the specific condition that most frequently triggers the evaluation short-circuit within the queue. Furthermore, beyond this queue positioning, the computational workload is inherently dictated by the physical nature of the situational conditions and the probability density of their fulfillment (or violation) across the simulated time horizon.

Conclusion and Future Work

This paper has presented a comprehensive description of software tools designed for modelling and solving situational tasks. Specifically, the parallel solver is tailored to handle large-scale problems by exploiting multi-core processors within shared-memory computing systems. To further optimize performance, a heuristic approach has been implemented, significantly enhancing computational efficiency when solving multi-condition situational tasks. Additionally, two distinct methods have been developed to improve the positioning accuracy for the entry and exit instants of the time intervals.

Future developments for the situational editor include introducing features for re-evaluating and modifying individual situational tasks, such as altering specific condition parameters or constraints, removing existing entries, and appending new ones. Regarding the parallel situational solver, subsequent work will focus on expanding the library of situational conditions through the formulation and programmatic implementation of new mathematical models.

References:

1. Wertz, J.R., Larson, W.J., Space Mission Analysis and Design. Microcosm Press, Kluwer Academic Publishers, third ed., 1999
2. Atanassov, A.M., Parallel satellite orbital situational problems solver for space missions design and control. *Advances in Space Research*, *58*(9), 2016, pp.1819–1826.
3. Atanassov, A.M., Situation Analysis Based on Spatiotemporal Situation Conditions, *Proceedings of SES.* 2016, 2017, pp. 47–51.
4. Atanassov, A., AN APPLICATION OF THE SITUATIONAL SOLVER TO PROBLEMS CONTAINING CONDITION RELATED TO VISIBILITY OF THE MOON BY A SATELLITE. *Proceedings of SES*, 2022, pp.62–67.
5. Atanassov, A., Applying a Situational Analysis Solver to Problems Involving Umbra/Sunlit Model. C. R. Acad. Bulg. Sci., v76, №7, 2023, pp. 1054 – 1064.
6. Atanassov A.M., APPLYING PARALLEL SITUATIONAL ANALYSIS SOLVER TO SATELLITE – SPACE DEBRIS CLOSE APPROACH PROBLEMS. ALGORITHMS AND SUBROUTINES. Aerospace Research in Bulgaria, 2025, 37, pp. 16–29.
7. Atanassov, A. M., ACCURATE DETERMINATION OF THE BOUNDARIES OF THE INTERVALS IN WHICH SITUATIONAL CONDITIONS ARE FULFILLED. ALGORITHM AND SUBROUTINES. Proceedings SES 2025, SRTI, BAS, ISSN:p-ISSN 2603 – 3313 e-ISSN 2603 – 3321, 2025, pp. 25–32
8. Atanassov, A.M., Efficiency Enhancement of Situational Analysis for Space Mission Design Using Reordered Heuristics, C. R. Acad. Bulg. Sci. v. 79(5), 2026, pp. 615–623.
9. Rauber, T., Rünger G., 2010. Parallel Programming: For Multicore and Cluster Systems. Springer.
10. Кальницкий, Л.А., Добротин, Д.А., Жевержеев, В.Ф., and Сапогов, Н.А., Специальный курс высшей математики для втузов, 1976.