SCHEDULE OPTIMISATION ALGORITHM FOR TRACKING SENSORS

J. A. Siminski(1), **T. Flohrer**(1), **and G. Sessler**(2)

(1)*Space Debris Office, ESA/ESOC, Darmstadt, Germany, Email: {jan.siminski,tim.flohrer}@esa.int*
(2)*Ground Station Systems, ESA/ESOC, Email: gunther.sessler@esa.int*

ABSTRACT

Tracking sensors, e.g. radar or laser ranging systems, are used to refine orbits of catalogued objects. A special tracking request has to be submitted whenever a reduction of uncertainty is required. This is the case for objects with a close approach above a certain risk level or whenever an object state gets uncertain to a level where it might drop out of the catalogue. However, if a sensor continuously runs to support a surveillance network to maintain a catalogue, an efficient track scheduling algorithm becomes important. As multiple objects can be seen simultaneously on the sky, decisions have to be taken which objects are observed and which are reachable afterward considering a limited steering velocity. This work proposes to formulate the track schedule optimisation as a longest path problem for a directed acyclic graph (DAG). Each track is modelled as a node of the graph and each edge connecting the nodes is assigned with a weight, e.g. information gain. The overall goal of the algorithm is to maximise the sum of these weights. It is achieved by finding the path through the graph with the largest overall weight. This weighted longest path optimisation problem can be solved in linear time for the special case of a DAG. The formulation of the problem and the solution algorithm will be presented along with an example schedule optimisation.

Keywords: scheduling, tracking sensors.

1. SCHEDULING PROBLEM

ESA, recognising the need to protect our critical infrastructure in space and on ground, has been undertaking a Space Situational Awareness Programme with three segments: Space Weather, Near Earth Objects and Space Surveillance and Tracking (SST) since 2009. It is intended to broaden the activities to address all aspects of space safety. Evolving SST towards Space Debris Monitoring, ESA will continue to address research and development of hardware and software technologies, ranging from sensors, sensor networking and data processing, applications, and standardisation. In this framework, tracking sensors are studied as means to provide high-accuracy data for objects involved in high-risk conjunctions and as a contribution to a sensor network. They are typically used to perform follow-up observations of objects that are already in the catalog or have been previously detected by a surveillance sensor. This means that a schedule for a tracking sensor contains predicted object passes instead of viewing directions.

Given a catalog with object states one can predict all visible passes for a defined time interval. The finite set of n observation opportunities is denoted with $(p_1, p_2, \dots, p_n)$. The first possible acquisition time of the pass p_i is denoted as $t_a(i)$ and the loss of signal time $t_e(i)$. Objects can be observed multiple times within the prediction interval.

Each observation pass provides an opportunity to update the respective object orbit. The priority of a pass can be determined by various factors [5, 4, 6, 3]. If the object is predicted to have a close approach with an operational satellite, it should be given higher priority. On the other hand, if an object state is already well determined because it has been recently seen by other sensors, it might require less observations from the tracking sensor. Another modification is the incorporation of likelihood of detection into the weights, as e.g. shown in [5]. If a close approach between two spacecraft without manoeuvre capabilities is detected, the participants must be tracked after the approach to discard a possible collision. The full radar pass of the object may be split into subsegments, as a single segment might provide enough new information to update the target and obtain a sufficiently small uncertainty estimate. For notational simplicity, a p_i pass can denote any kind of segment where an object can be observed. The overall importance of the pass is defined with a weighting function w.

The tracking schedule optimisation objective function is then defined with

$$F(\mathbf{s}) = \sum_i w(s_i, \mathbf{s}) \tag{1}$$

where the sequence or order of scheduled object passes is

$$\mathbf{s} = (s_1, s_2, \dots) \quad s_i \in (1, \dots n) \tag{2}$$

and the sequence is subject to the following constraint

$$t_a(s_{i+1}) \geq t_e(s_i) + \Delta t(s_i, s_{i+1})\,, \tag{3}$$

and

$$\Delta t(i,j) \quad \text{for} \quad i,j \in (1,\dots n) \tag{4}$$

describes the time needed to point the sensor from the state at the end p_i to the first state of pass p_j. The weighting function w is not defined for a single pass s_i, but the whole sequence. If an object appears multiple times in the list of passes, the benefit of an observation can differ depending on whether the object has already been observed or not.

The sequence **s** of unknown length, which maximises the objective function in equation (1), represents the optimal schedule for the tracking sensor, i.e. $\mathbf{s}_{\text{opt}} = \arg\max F(\mathbf{s})$. This sum maximisation in its general form is considered as a *NP-hard* permutation problem, where the computation time grows exponentially with the number of predicted passes n. Typically, these problems are tackled with heuristics or approximate algorithms [8].

2. DIRECTED ACYCLIC GRAPHS

The scheduling optimisation can be rewritten as a longest path problem for a directed acyclic graph (DAG) $G = (V, E, W)$ assuming certain simplifications. The usage of DAGs for sensor schedule optimisation has already been demonstrated e.g. in [1]. A DAG is an ordered structure containing a finite set of edges E and nodes V (also called vertices). Each pass p_i is a node in $V = (p_1, \dots, p_n)$. The edges $E = (e_1, \dots, e_m)$ are constructed to connect the passes. They represent the movement or control from one pass to the next. The weights $W = (w_1, \dots, w_m)$ describe the length of each edge. An edge is created between p_i and p_j if the constraint from equation (3) if fulfilled, i.e.

$$c(i,j) = \{i,j : t_a(j) \geq t_e(i) + \Delta t(i,j)\}\,. \tag{5}$$

The graph is acyclic (it has no directed circuits) as we cannot go back in time to observe an object and then reach the current state again.

If a pass p_j exists, such that $c(i,j)$ and $c(j,k)$ is fulfilled, the edge between i and j is unnecessary as the sequence (i,j,k) will always have a longer path than (i,k) assuming positive edge weights only. The latter assumption is valid, as observation passes can always be considered as beneficial (positive weight) for the cataloguing and orbit determination process. This assumption helps to reduce the number outgoing edges of p_i and thus the overall number of edges m. Figure 1 illustrates the reduction of edges.

The path through the graph which maximises the sum of weights along the edges is considered the longest. This task is called longest path or critical path problem and is very common in computer science or project management for job or task scheduling. For a general graph, the solution is complex and considered an *NP-hard* problem as well. However, for a DAG the solution can be found in linear time, i.e. the algorithm terminates after a number of steps in the order of $\mathcal{O}(n+m)$. The solution of the longest path problem is equivalent to the maximisation

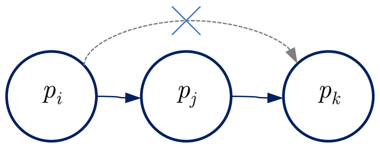

Figure 1. Removal of unnecessary edges in planning graph.

of the objective function in (1), however, assuming that the weights are independent of the selected sequence of passes **s**. The inclusion of dependencies in the graph is in principle possible by branching of subgraphs. The inclusion and its limitations will be discussed later in this paper.

2.1. Algorithm

The algorithm to find the longest path has two phases. First, the so-called acyclic ordering is found by sorting the graph (called topological sorting). When predicting the satellite passes, and filtering out feasible connections between passes, this order can be directly established, i.e. for every edge between p_i and p_j from the sorted list of passes, it is guaranteed that $i < j$. The reader is referred to textbooks such as [2, 9] for an implementation of topological sorting.

The longest path is then found as described in [2] and summarised in the following. The longest path ending in each node is computed by looking at all nodes of the incoming paths. The values computed for the nodes of the incoming paths are re-used. If no incoming paths exists, the vertices path is initialised with 0. The algorithm gains its efficiency due to the re-use of already computed quantities (a method typically called dynamic programming).

2.2. Conditional weights

In order to account for conditional weights (different weights depending on path), the graph can be extended with a subgraph. The typical scenario would be a re-observation of an object. Assuming that an object can be observed multiple times, it might be favourable to down-weight a pass if it is already observed at another time. The subgraph would be created at each observation of the object and then duplicate the nodes of the main graph until re-observation. It would, however, then consider a different weight for all edges pointing towards the next pass of the object.

The number of combinations grows exponentially with the number of multiple-times observed objects. Hence,

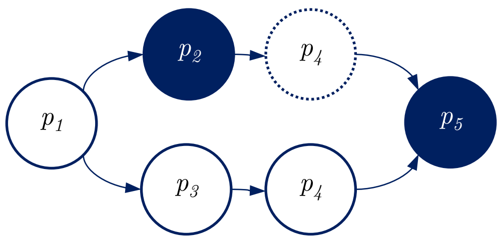

Figure 2. Illustration of a new branch to account for weight dependency. The pass p_2 and p_5 are both generated by the same object. The node p_4 must be duplicated in order to allow different edge weights depending on the path.

the direct solution of it is not feasible and heuristics must be used instead.

A simple and straight-forward way of implementing conditional weights, is to split the prediction time interval into segments with little or no re-observations of the same objects (e.g. around 100 minutes). The path finding is then performed over the first segment only and the weights can be recomputed for the later segments. This solution is suboptimal as a later pass of an object could provide more information, but is discarded as the previous one was already selected. Another simple approach is to find the optimal path over the whole prediction time. For all multiple times observed objects, select the pass along the optimal path with the largest weight and fix it, i.e. remove all edges bypassing this vertex. Then recompute weights for the other passes of the objects and repeat the path finding over the whole arc. Lastly, evolutionary algorithms (e.g. as described in [8]) can be used to find better global solutions. The algorithms typically have to be initialised with a guess. Further research should assess how much these heuristics approaches can approximate the optimal solution of the sensor scheduling problem.

2.3. High-priority pass requests

A high-priority pass, e.g. of an object which is about to have a close encounter, is selected by splitting the graph at the node. As already described in the section above, all edges can be removed that bypass a specific node. This way it is guaranteed that the longest path will traverse through the requested point. An illustration of such a request is shown in figure 3.

2.4. Pass splitting

A special type of re-observation happens when an object pass is split into pass segments. As written in the introduction, this can help to increase the number of observed objects. If the information content of a short arc is sufficient for an orbit update, the sensor should prefer updating another target over completing the full pass. If too many subsegments are created, the problem becomes unfeasible to solve due to the amount of possible combinations. Heuristics approaches could again be used to find an approximation to the optimal solution. However, splitting a pass in two segments can be easily incorporated into the DAG. The weight of the edge pointing from one segment to the other can be reduced to account for the already gained information. The edges connecting other target passes to the second segment should thus be larger. The exact weight computation is outside the scope of this paper. Typically, information gain (or improvement of covariance) is used to quantify the benefit of an observation [6, 3].

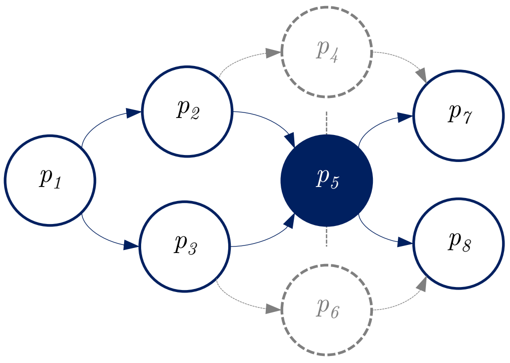

Figure 3. Illustration of a high-priority request: the pass p_5 is used to split the graph, removing all edges bypassing it and all other simultaneous passes.

3. SENSOR MOTION MODEL

The Δt function in equation (4) describes the time needed for the sensor to point from one state to the other. The state of the system is represented with two angles (e.g. azimuth and elevation) and their time derivative (angular rates). The sensor pointing is modelled with the equation of motion

$$\dot{\phi} = f(t, \phi, u)\,, \tag{6}$$

where ϕ is one of the angles and u describes the control, e.g. servo-motor acceleration. For sake of simplicity, the equation is only shown for one angle, but it could be extended to multiple angles.

If a linear motion model is assumed with instantaneous acceleration and deceleration, the minimum time between two pointing directions can be computed with

$$\Delta t(i,j) \leq \frac{(\phi_{a,j} - \phi_{e,i})}{\dot{\phi}_{\text{max}}} \tag{7}$$

where $\dot{\phi}_{\text{max}}$ is the maximum slew velocity of the sensor.

A more advanced logic has been implemented, which in addition to a maximum slew velocity also considers a maximum acceleration. Given the initial conditions $(\phi_i, \dot{\phi}_i)$, the time-optimal control solution to reach the final state $(\phi_j, \dot{\phi}_j)$ can be analytically computed. The optimal position, velocity and acceleration profile for the boundary value problem are shown in figure 4. The minimum-time $\Delta t(i, j)$ is then used to find out if both passes can be connected or not. The time can differ depending on the selected observation strategy. One can either reach the next target at a specified velocity to start blind tracking (same pace as next target). Alternatively, the sensor could wait for the target to appear (possibly scanning a certain region above the horizon) and start tracking once the object appears in the beam.

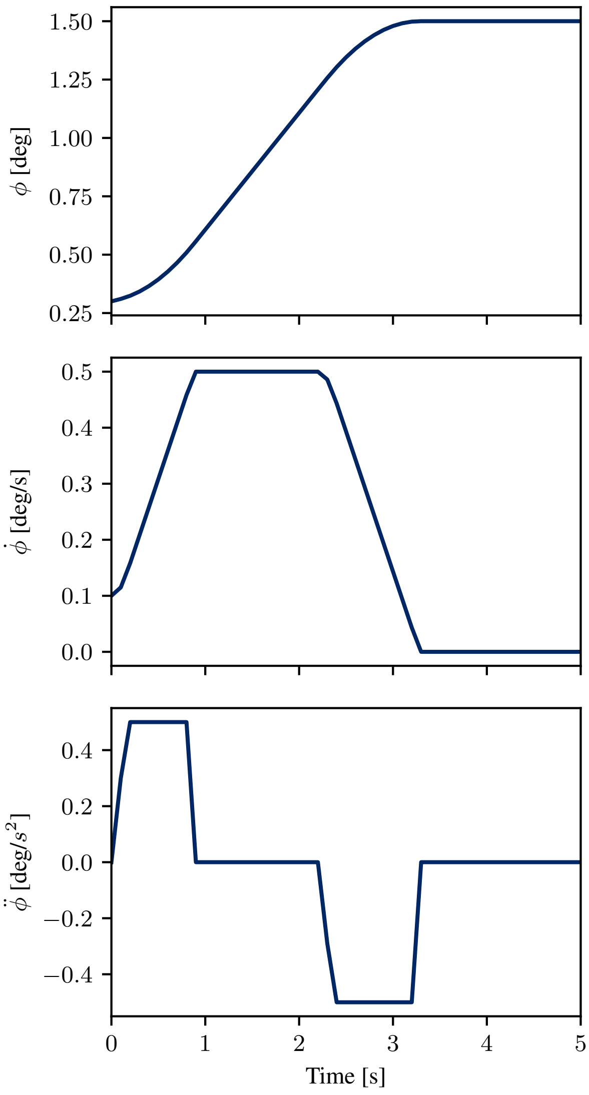

Figure 4. Angular position, velocity, and acceleration profile of the time-optimal control solution reaching the state $(\phi_j, \dot{\phi}_j) = (1.5, 0)$ *(deg and deg/s) starting from* $(\phi_i, \dot{\phi}_i) = (0.3, 0.1)$

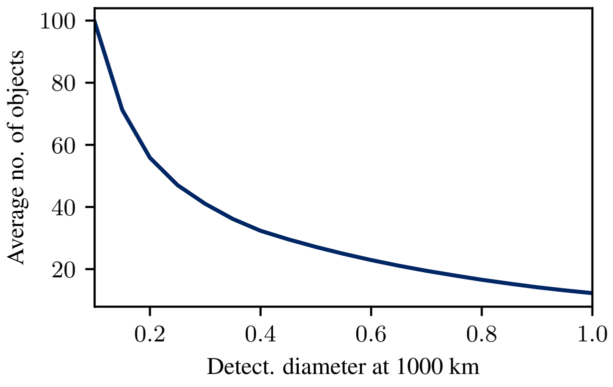

Figure 5. Average number of simultaneously detectable objects depending on detectable object diameter at 1000 km distance.

4. TEST CASE

A tracking radar schedule is created in order to test the algorithm and assess the runtime performance. The fictitious radar is located in central Europe, and its detection performance is varied to observe how the performance scales with the number of trackable objects. The simulated radar setup is described in table 1.

Table 1. Tracking radar setup

Parameter	Value
Longitude	8.6°
Latitude	49.8°
Height	0 km
Min. elevation	20°
Detect. diameter at 1000 km	$0.1 - 1$ m
$\dot{\phi}_{\max}$	1 deg/s

Around 12000 low-Earth objects are propagated over a time frame of 6 hours. Two-line elements from `www.space-track.org` are used to initialise the object states. The object diameters are extracted from the DISCOS database [7]. An object is considered detected, if the diameter is above the detection diameter at the observed range. The simulation setup is equivalent to the setup explained in [10] (detection method is explained in more detail there). In addition to a detection threshold, the maximum tracking velocity is used to filter out observations that cannot be reached by the sensor. The passes are not segmented for this simulation.

The resulting average number of objects visible at any instance in the sky is shown in figure 5. The number of simultaneously visible objects is important as it determines the width of the graph and increases the complexity of the optimization problem.

The number of objects that have been observed multiple times, increases the optimisation efforts as weights have to reassessed. Figure 6 shows that this needs to be considered for prediction times over around 100 min. Branching off graphs to account for changed weights becomes unfeasible afterward and the heuristics as described above must be used. The prediction time should be selected based on possible feedback from a central scheduler in a surveillance network, e.g. once new predictions are available or new high-priority targets have been identified.

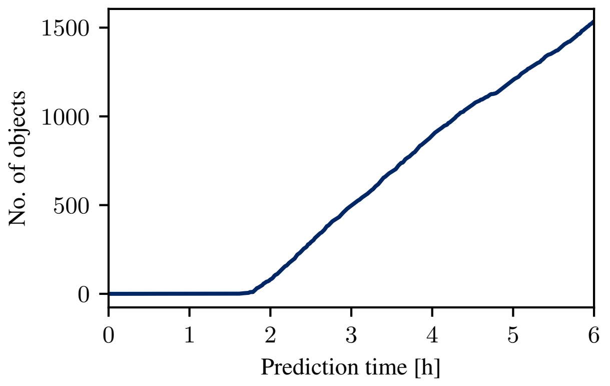

Figure 6. Number of multiple times observed objects for a radar that can detect an object with diameter 0.1 m at 1000 km distance.

A simple weighting scheme is used for the demonstration in this paper. Each pass is initially weighted with 1. If the longest path contains multiple passes of the same object, the first one is selected and all others nodes are weighted with 0. Afterward the longest path is recomputed. The result of the optimisation gives a longest path with 160 tracks and no object is re-observed.

Figure 7 shows the required runtime for different tested radar performance values. The computation is performed on laptop with 2.9 GHz Intel Core i5 CPU and 8 GB RAM using the publicly available Python library *pygraph*. The runtime is negligible in comparison with the time to predict the satellite passes (around 20 minutes). The latter has to be computed for any kind of schedule optimisation and is therefore not considered in this analysis. In an operational environment, the satellite orbits might be propagated at the cataloguer level. They would be then also used for close approach detection and other services.

The runtime is coupled with the number of edges and nodes. Figure 8 shows the dependency of the size of the graph on the detection capabilities of the radar. The figures also illustrate the dependence of runtime and graph size on the number of simultaneously observed objects. It should be noted that the prediction time scales the runtime linearly as the graph width is not increased but just its length.

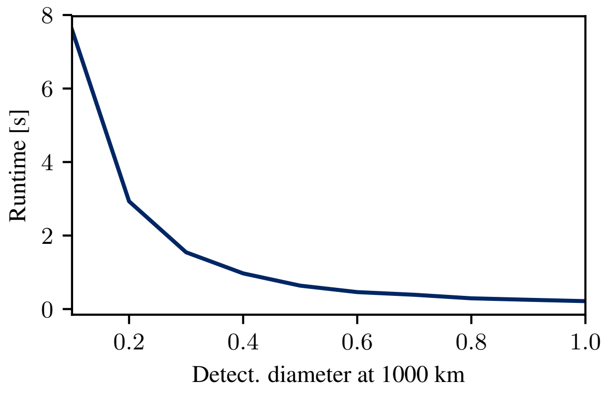

Figure 7. Runtime depending on detected diameter at 1000 km distance.

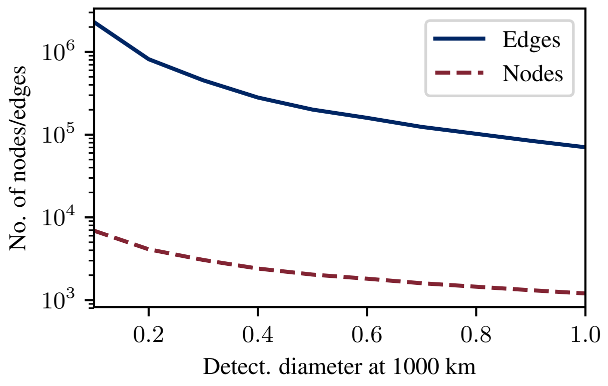

Figure 8. Number of graph edges and nodes depending on detected diameter at 1000 km distance.

5. CONCLUSION

When structuring the optimal scheduling problem for a tracking sensor as a DAG, it reduces to a simple longest path search and can be solved with negligible computation time. DAGs offer a dynamic framework for the scheduling of a sensor as part of a surveillance system. Tracking requests for high-priority targets can be easily included in the DAG to generate an updated plan.

Dynamic weights can be included, however, requiring then a heuristic approach, i.e. the longest path is iteratively found while updating the dependent weights. The method is shown to be efficient. More challenging cases need to be tested using realistic weights, e.g. considering the information gain. The results can then be compared with solutions obtained with evolutionary algorithms to assess how well the objective function is optimised.

REFERENCES

1. Augenstein, S. (2014). Optimal scheduling of earth-imaging satellites with human collaboration via directed acyclic graphs. *Presented at AAAI Spring Symposium on the Intersection of Rosbust Intelligence and Trust in Autonomous Systems*, Stanford, California, USA.
2. Bang-Jensen, J., Gutin, G. Z. (2008). Digraphs: theory, algorithms and applications. Springer Science & Business Media.
3. DeMars, K. J., Jah, M. K. (2011). Evaluation of the information content of observations with application to sensor management for orbit determination. Presented at AAS/AIAA Astrodynamics Specialist Conference, Girdwood, Alaska, USA.
4. Früh, C., Fielder, H., Herzog, J. (2017). Heuristic and Optimized Sensor Tasking Observation Strategies with Exemplification for Geosynchronous Objects. *Journal of Guidance, Control, and Dynamics*, **41**(5), 1036–1048.
5. Gehly, S., Bennett, J. (2016). Incorporating Target Priorities in the Sensor Tasking Reward Function. *Presented at the Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS)*, Maui, Hawaii, USA.
6. Hinze, A., Fiedler, H., Schildknecht, T. (2016). Optimal scheduling for geosynchronous space object follow-up observations using a genetic algorithm. *Presented at the Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS)*, Maui, Hawaii, USA.
7. McLean, F., Lemmens, S., Funke, Q., Braun, V. (2017). DISCOS 3: An improved data model for ESA's database and information system characterising objects in space. *Presented at the 7th European Conference on Space Debris*, Darmstadt, Germany.
8. Price, K. V., Storn, R. M., Lampinen, J. A. (2005). Differential evolution: a practical approach to global optimization. Springer-Verlag Berlin Heidelberg.
9. Sedgewick, R., Wayne, K. (2011). Algorithms. Addison-Wesley Professional.
10. Siminski, J.A. (2016). Techniques for assessing space object cataloguing performance during design of surveillance systems. *Presented at 6th International Conference on Atrodynamics Tools and Techniques (ICATT)*, Darmstadt, Germany.