Deutscher Luft- und Raumfahrtkongress 2009
DocumentID: 121180

A PLANNING SYSTEM FOR AUTOMATED ALLOCATION OF ESA GROUND STATION NETWORK SERVICES

Alexander Hoffmann
Holger Dreihahn
Marc Niézette
VEGA Deutschland GmbH & Co. KG
Europaplatz 5, D-64293 Darmstadt, Germany

Gerhard Theis
European Space Agency
Robert Bosch Straße 5, D-64293 Darmstadt, Germany

ABSTRACT

The **ESTRACK Planning System** (EPS) is a fully integrated planning system dedicated to the automated centralized allocation of ground station services to space missions. The EPS is operationally used at the European Space Operations Centre in Darmstadt, Germany. Instead of assigning a ground station statically to a mission, the EPS identifies ground stations which can provide the required services at the required times to a mission and plans the ground station allocation for all participating missions. The system is flexible enough to incorporate new missions and ground stations into planning process by simply changing the software's configuration database. The inputs to the EPS are contact requirements of the client missions, and event files containing amongst others the visibility patterns between spacecrafts and ground stations. This static information is to be augmented during EPS operations by dynamic refinement requests updating parts of the original contact requirements. The control is performed by an operator which creates and commits contact allocation plans. Produced plans are stored in the **ESTRACK Management Plan** (EMP), which consists of a set of booking periods of the ground stations by the missions with the associated required services. This booking evolves as the event timings are updated and as more and more missions are taken into account. The EMP is then used by the **ESTRACK Scheduling System** (ESS) to generate executable ground station schedules.

1 EPS OVERVIEW

The European Space Agency runs a number of ground stations to support its own and external, e.g. NASA, missions. 8 ESA's own stations plus 3 co-operative stations form the basis of the *ESA Tracking Network* (ESTRACK), which also includes control and communication facilities. ESTRACK currently supports 10 operational ESA science missions, providing services for data downlink and the uplink of commands to satellites in orbit.
In the past, planning and scheduling of the ESTRACK were done manually by operators, supported by a set of tools, however without automated conflict resolution or intelligent search algorithms. Clearly, this approach depends very much on human intervention. In order to alleviate this dependency and to coordinate the growing number of missions and ground stations efficiently, an automated planning and scheduling system was developed. The capability to optimize the space and ground resources usage, to perform tasks more quickly and to manage complex configuration control tasks are the main advantages of the automated planning and scheduling features of the *ESTRACK Management Systems* (EMS). EMS is a part of the *ESA Ground Operations Software* initiative, which includes software systems covering all relevant ground systems of a space mission, see www.egos.esa.int. Figure 1 shows the EMS in its operational context.

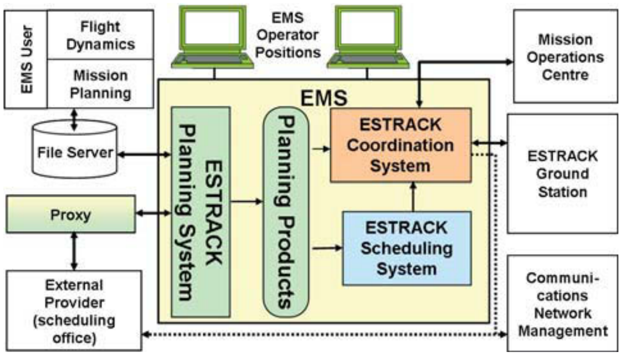
EMS User
Flight Dynamics
Mission Planning
EMS Operator Positions
Mission Operations Centre
EMS
File Server
ESTRACK Planning System
Planning Products
ESTRACK Coordination System
ESTRACK Ground Station
Proxy
ESTRACK Scheduling System
External Provider (scheduling office)
Communi-cations Network Management

Figure 1: EMS Operational Context

ESTRACK Planning System is one of the three elements of the EMS, see Figure 2. As the name suggests, EPS is responsible for the planning activities within the EMS. Input data to the EPS are contact requirements of the space missions, and event files containing amongst others the visibility patterns between spacecrafts and ground stations coming from flight dynamic departments. The EPS design is based on a layered planning approach as depicted on Figure 2.

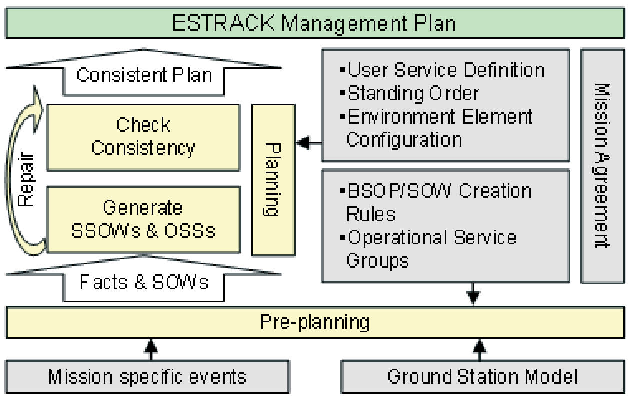

Figure 2: EPS Layered Structure

2 PRE-PLANNING

Missions' event files provided by the flight dynamic departments and service requirements configured in each mission agreement will be processed during the pre-planning phase. Outputs of the pre-planning are facts, *Service Opportunity Windows* (SOW) and *Basic Standing Order Periods* (BSOP). These elements constitute the input for the following planning sessions. The only reason for separating the pre-planning and planning sessions, is that the processing of large event files can be quite time consuming but does not require any interaction with an operator. This decoupling allows for example to run a pre-planning session as a batch job over night.

2.1 Fact

Facts represent temporal states of environment elements. An environment element can represent a ground station, an operator shift, some orbit event or some other event which has to be taken into account during the planning session. Two events of an event file can be combined to a fact. Facts can also be created from a single event to support the import of single events from event files. Each fact has start and end times derived from events and a state, for example a fact can have a state called *visibility* and start and end time points corresponding to the ones of a particular visibility window.

2.2 Basic Standing Order Period

A BSOP is a period of time during which a service specified in the User Service of a mission agreement must be provided. BSOP is defined in the Standing Order inside the User Service. Roughly speaking, services required by a mission agreement have to be provided based on the interval given by the Standing Order. This interval can be based on mission specific events, e.g. an orbit event, or on the Gregorian calendar, e.g. 2 days or 1 week. The interval expressed by a BSOP is called 'basic' because service implementation does not necessarily have to happen in every BSOP. Services can be required every n^{th} BSOP. The BSOP generation works in steps:

- Parsing events from event files;
- Generation of facts from parsed events;
- Generation of BSOPs from one or more facts by application of BSOP creation rules.

2.3 Service Opportunity Window

The missions participating in the EPS planning process feed on a regular basis their predicted events into the EPS. In that context, predicted events are:

- Acquisition and Loss Of Signal events for the satellite-ground station combinations of a mission;
- Start and end of operator shifts;
- All other events relevant to planning of ground station allocation.

According to mission specific rules, the predicted events will be combined to SOWs. A SOW is a period of time, during which a ground station can provide the set of required services. The SOW generation works in steps:

- Parsing of events from event files;
- Generation of facts from parsed events;
- Generation of SOWs from one or more facts by application of SOW creation rules;
- The generated SOWs will be filtered by dropping those SOWs whose ground station cannot provide the set of required services.

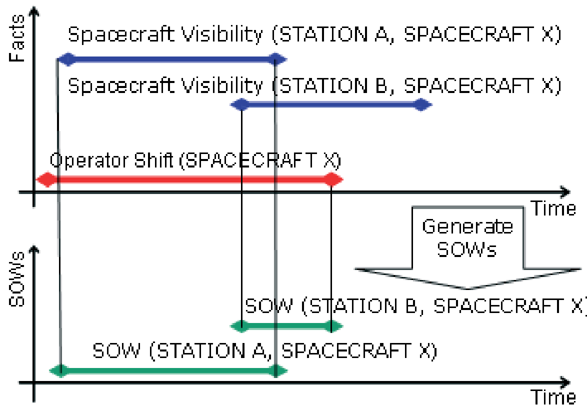
Facts
Spacecraft Visibility (STATION A, SPACECRAFT X)
Spacecraft Visibility (STATION B, SPACECRAFT X)
Operator Shift (SPACECRAFT X)
Time
Generate SOWs
SOWs
SOW (STATION B, SPACECRAFT X)
SOW (STATION A, SPACECRAFT X)
Time

Figure 3: SOW Generation

SOWs will be created by so called SOW rules, which are statements formulated in the *Language for Mission Planning* (LMP). LMP is a query and rule language designed for processing temporal objects stored in a database and creating new objects from the queried ones. SOW generation rules are part of the mission agreement of each mission and are associated to User Services. This allows individual SOW generation per User Service. LMP expressions denote the required facts, like operator shift and spacecraft visibility, and their temporal relationships. Here is an example for an LMP expression:

$$\begin{aligned}&fact\,(?id, ?gs, state, ?vStart, ?vEnd)\\ &\wedge\, parameter\,(?id, satelliteId, ENV)\\ &\wedge\, ?gs\; = \;".*erth"\\ &\rightarrow\, activity\,(?newId, sowGenerator, SOW, ?start, ?end)\\ &\rightarrow\, parameter\,(?newId, groundStation, ?gs)\end{aligned}$$

The first two lines of the expression filter from the database all facts with a particular state, namely *state*, which have a parameter *satelliteId* set to *ENV*. While evaluating the first line, the variable *?gs* was instantiated with the name of the ground station, which is also the fact name. In the third line the fact name contained in *?gs* is tested against the regular expression *.*erth*, which matches for example the string *Perth* (location of a ground station). Once all conditions of the SOW rule in the lines 1 - 3 evaluate to true, the actions in the lines 4 and 5 are performed. The fourth line creates a new activity representing a SOW and instantiates the variable *?newId* with the ID of the created activity. The last line sets parameter *groundStation* for the created activity (identified by the instantiated *?newId* variable) to the value held in variable *?gs*, instantiated in the first line. For more details on LMP please refer to [1].

3 PLANNING

The aim of the planning session is to produce a valid plan implementing the mission agreements for all missions in a finite time range. Planning results are stored on the ESTRACK Management Plan as so called Operational Service Sessions.

- An *Operational Service Session* (OSS) is a basic component of an EMS plan, which represents a group of operational service instances with their respective execution timings, whose execution involves a single ground station, together with relevant communication networks.

- A *Candidate Operational Service Session* (COSS) is an OSS whose start and end times are time variables. The following attributes are associated to each COSS: a parent SSOW, a supporting SOW, position in the SSOW, and variable start and end times.

- A *Super Service Opportunity Window* (SSOW) is a time period in which a *continuous* service provisioning within a particular BSOP can happen. SSOW is composed of a sequence of overlapping SOWs and their associated COSSes. The following attributes are associated to each SSOW: related BSOP, start and end dates, a set of used SOWs, a sequence of COSSes sorted by increasing start date.

- In the mission agreement model, services requested by a particular mission are gathered in an *Operational Service Group* (OSG). Several OSGs can be associated to a User Service Definition, which defines one of two mutually exclusive user service levels, i.e. nominal or degraded.

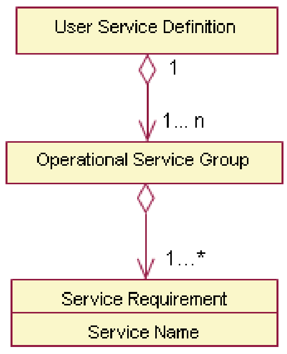

Figure 4: User Service Definition

In general it can be said, that a SSOW is required for each OSG implementation. This implies that the number of required SSOWs per mission and BSOP is the number of implemented OSGs multiplied by the required service repetitions of each OSG. Only user service definitions which require more service sessions at the same time on different ground stations will have more than one OSG.
Based on the facts, SOWs and BSOPs prepared by the pre-planning, the EPS planning process tries to assign to each selected BSOP a set of OSSes implemented on SOWs such that all requirements in the mission agreement and all resource constraints are respected.
A BSOP is considered to be planned if a set of COSSes has been generated that implements the associated User Service within the corresponding time slot. The start and end times of a set of COSSes constitute a set of variables of a temporal constraint network. The domains of these variables are determined by the start and end times of the supported SOWs, while the constraints between those variables are provided by the corresponding User Service and inferred from the used resources. Given that, a valid plan on a set of planned BSOPs can be generated if and only if the underlying temporal constraint network is consistent. If a given constraint network has been proven to be consistent, then COSSes, which constitute the network, will be promoted to OSSes by the simplex algorithm. The simplex algorithm fixes the start and end points of COSSes ensuring at the same time the BSOP-local optimality of the plan. The objective function for the BSOP-local optimality is defined as a sum of OSS durations for a space craft on a particular ground station multiplied by the priority of using this ground station by the space craft, i.e.

$$\sum_{OSSes} P\,(sc, gs)\cdot\left(OSS_{end}^{(sc,gs)} - OSS_{start}^{(sc,gs)}\right) \tag{1}$$

where gs denotes a ground station, sc stands for a space craft and $P\,(sc, gs)$ is the corresponding priority. Therefore, the general planning problem will be decomposed into two basic problems:

- generation of the COSSes for each BSOP;
- consistency checking of the underlying temporal constraint network.

The former can be modeled as a selection or planning problem and the latter as a scheduling problem.
As already mentioned, the planning process is guided by the configurable priorities for pairs of ground stations and spacecrafts, which allows the user to configure the EPS to prefer certain ground stations for a particular spacecraft and controls the way conflicting usages of ground stations are resolved. In the case of a conflict, the set of so far generated COSSes has to be changed. The EPS supports automated repair, degradation of service requirements, and finally provides useful conflict information helping operators to manually eliminate conflicts. Finally, when the plan is completed for the specified planning period the operator can produce graphical or textual plan views to inspect the plan. If the plan is alright and passes the inspection, the plan can be committed to the EMP.
If a created plan needs to be changed, e.g. event timings have to be updated or more missions have to be taken into account, the operator simply starts a new planning session (or first pre-planning and than planning) with new requirements. When trying to implement new requirements on an existent plan, it is desirable to fix the timings of the previously planned events. This can be achieved by assigning so called *plan stability constraints* to each event time point. For example, in order to fix the start time point of an event A one has to issue two plan stability constraints

(2) $$A_{start} \leq a \wedge A_{start} \geq a$$

The plan stability constraints will be applied to a consistent plan. Hence, in the case if some of them cause a conflict, they can be simply removed in order to ensure plan consistency at the price of changed timings of already planned events.
The following state diagram shows the general structure of the EPS planning algorithm. Note that two options were available BSOPs implementation:

- either plan all the BSOPs, then check the consistency of the global underlying constraint network, and perform some repairs if necessary;
- or plan one new BSOP, check the consistency of the underlying constraint network, performing a repair if necessary, then plan the following BSOP, and so on (incremental approach).

We chose the incremental approach essentially in order to make the repairs easier. Thus, at each pass in step "Select an unplanned BSOP" of the general algorithm, a BSOP is heuristically picked and removed from the set of unplanned ones. In our implementation, the BSOPs are ordered by increasing end time and put in a queue (earliest deadline first heuristic), with the hope to limit the extent of the possible repairs to the near past. Because of the repair procedure the

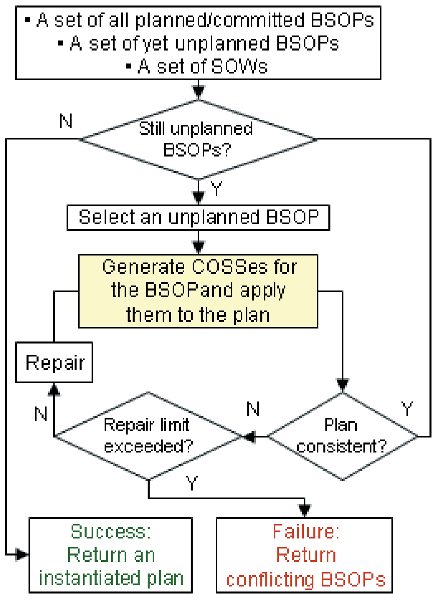
▪ A set of all planned/committed BSOPs
▪ A set of yet unplanned BSOPs
▪ A set of SOWs
Still unplanned BSOPs?
N
Y
Select an unplanned BSOP
Generate COSSes for the BSOPand apply them to the plan
Repair
Repair limit exceeded?
N
Y
Plan consistent?
N
Y
Success: Return an instantiated plan
Failure: Return conflicting BSOPs

Figure 5: Planning Algorithm

overall algorithm is not complete. It is possible that it returns with a failure status while a valid plan actually exists.
It is worth to note that in addition to the standard way to change a plan by planning additional missions, EPS provides features and tools to "manually" edit a plan. It is capable to process standing order and OSS refinement files. Standing order refinement files contain service requirement changes for a particular set of BSOPs. For example, a desirable number of passes can be changed or the contact handover between different ground stations can be forbidden for a particular space craft. The OSS refinement files are applied to a given plan in order to commit, delete or add an OSS or to modify an OSS in terms of time or configuration profile.

4 PLANNING STRATEGIES

Now, we take a closer look at the module that generates a set of COSSes for a chosen BSOP (highlighted yellowish on the Figure 5). Currently, depending on the kind of mission agreement, EPS uses two different approaches for generation of COSSes. The first approach is based on the *dynamic programming* [2] and suitable for mission agreements with

- small number of SOWs in each BSOP;
- small number of required service repetitions;
- "flexible" duration constraints, i.e. maximum and minimum service durations are not equal.

The second approach is based on the *local search*. It performs better than the one based on the dynamic programming if either the number of SOWs in each BSOP or required service repetitions is high or a given mission agreement has "inflexible" duration constraints, i.e. maximum and minimum service durations are equal. Moreover, the local search approach must be used in the case when at least two Operational Service Groups must be planned per BSOP, and when they are not constrained to occur in sequence, i.e. there is no distance constraint in between. For constellation of satellites, e.g. Claster and Herschel-Planck, EPS uses solely the local search planning strategy.

4.1 Dynamic Programming

Assembling SSOWs for a BSOP, we consider the given BSOP as a sequence of decision instants defined by the start and end times of each SOW in the BSOP. At each instant, we can decide to keep contact with the current SOW or perform a handover to another available SOW at this time. A SSOW is then a finite sequence of such decisions:

- keep the contact with this SOW until the next decision instant;
- choose an available SOW and contact it.

An example for generation of a SSOW, which is a set of COSSes, is depicted on the Figure 6.

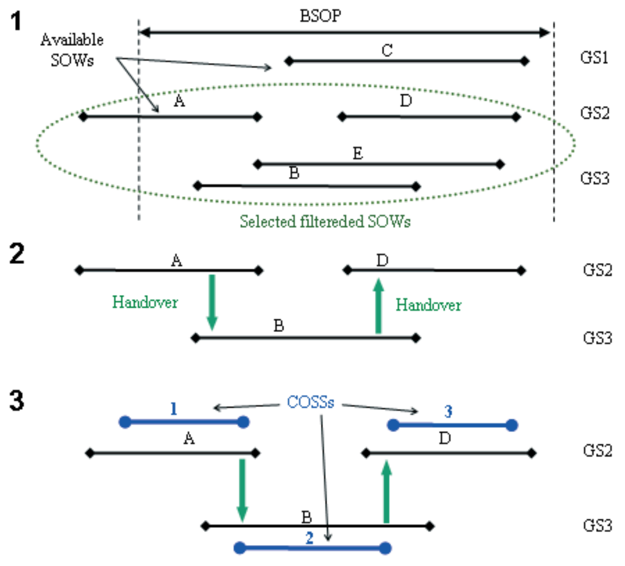

Figure 6: SSOW Generation

This is a first order Markovian sequential decision problem, where decisions have to be made in successive steps. A decision at one step depends only on the one from the previous step. Decisions are made taking into account only ground station preferences and the requirement of maximum possible contact duration with the ground station. For the modeling purpose, a SOW s has:

- a start time s_{start};
- an end time s_{end};
- a priority $P(s)$ taken from the priority table for pairs "ground station - spacecraft".

The gain of accessing to s for a duration T can be defined as

(3) $$G(s,T) = (T_{end} - T_{start}) \cdot P(s).$$

Let S be the set of so far available SOWs.

- Steps: one for each start and end times of the SOWs of the BSOP
- States: contact with one of the SOWs from current time to the next step; the set of possible states at step i is
  (4) $$X_i = \{x \in S \mid t_i \in [x_{start}, x_{end}) \wedge t_{i+1} \leq x_{end}\}$$
- Actions: stay connected to the current SOW until next time step, then contact an available SOW taken from the set X_{i+1} or stop the SSOW.
- The reward function for the action that passes from $x \in X_i$ to $y \in X_{i+1} \cup E_{i+1}$

  (5) $$v(x,y) = G(y, t_{i+2} - t_{i+1})$$

  where $E_i = \{z \in S \mid z_{end} = t_{i+1}\}$ is the set of following possible states to which no more action can be applied. These states terminate the SSOW.
- Preference relation on the rewards: $\geq$

The optimal gain for each step i and each state $x \in S$, denoted $V_i(x)$, is the maximal reward that can be obtained in state x at time t_i. Initial conditions:

(6) $$\forall i \ \ \forall x \in S \quad V_i(x) = \begin{cases} 0 & \text{if } t_i = x_{start} \\ -\infty & \text{else} \end{cases}$$

The Bellman equation allows us to compute the optimal gains:

(7) $$V_i(x) = \max_{\tilde{x} \in X_{i-1}} \{v(\tilde{x}, x) + V_{i-1}(\tilde{x})\}$$

The SOW that provides the best reward if the spacecraft uses it at time t_{i-1} and uses the SOW x at time t_i is

(8) $$prev_i(x) = \arg\max_{\tilde{x} \in X_{i-1}} \{v(\tilde{x}, x) + V_{i-1}(\tilde{x})\}$$

If $X_{i-1} = \emptyset$, then $prev_i(x) = null$. When all the partial optimal gains $V_i(x)$ are computed, we determine the best global reward, that is

(9) $$V = \max_S V_i(x)$$

We pick up the associated step i and state x and build the optimal plan from the last step to the first using the $prev_i(x)$.
This approach is sound, which means that on success returned plan is valid. The worst case time complexity of this algorithm is $O(|S|^3)$.

4.2 Local Search

The second strategy implemented in the EPS to generate the SSOW is the local search algorithm, which is intended for elimination of drawbacks due to complexity of the dynamic programming. The main idea of this approach is to generate SSOWs randomly and assign them values. Eventually, a solution with the best value will be chosen. Looking for a solution with the best

value we start from a candidate solution and then iteratively move to a neighbor solution. The search space in our case consists of all possible sets of SSOWs in a BSOP. As we already mentioned, a set of SSOWs can be seen as a solution only if it has a SSOW associated to each service repetition of each OSG. In order to implement the local search planning strategy on a set of SSOWs, we have to define a neighborhood relation on this set. We say that two sets of SSOWs are neighbors if they differ only in one SSOW. Having found a solution, we compute its value by running simplex algorithm on the underlying constraint network, and then we try to replan a randomly chosen SSOW ensuring that it consist of different set of SOWs. The SOWs, which are forbidden for the next pass, will be stored in a tabu list.
Note that the dynamic programming used for SSOW generation is a deterministic algorithm, which means that it will always give the same result for given time bounds and forbidden SOWs. Hence, the degrees of freedom that one has when trying to plan all SSOWs are only the following:

- the choice of the SSOW to plan/unplan next;
- the SOWs that are forbidden when planning a SSOW.

Another degree of freedom is the number of service repetitions, when several values are allowed.
The local search approach requires a termination criterion in order to avoid an infinite loop caused by its random behavior. Hence, the local search algorithm is sound but not complete, in contrast to the dynamic programming approach, which is sound and complete. Figure 7 depicts the transition from one solution to another for an example where only one service repetition for each of two OSGs, blue and green, is required.

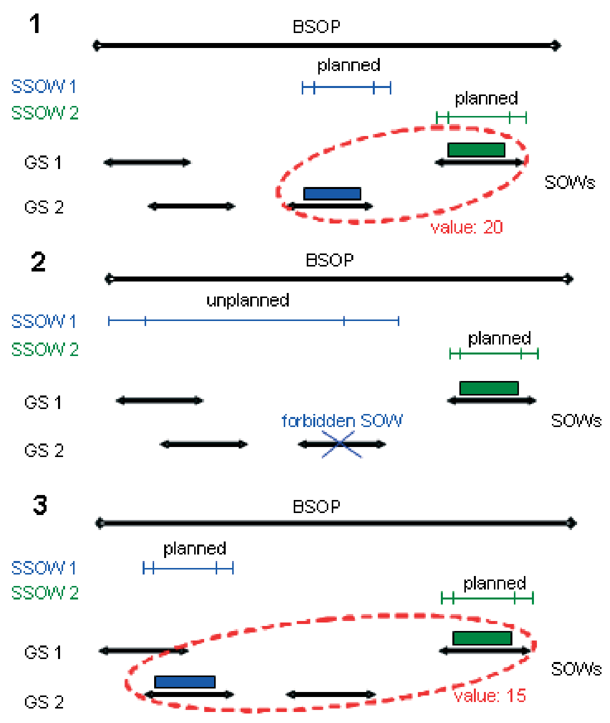

Figure 7: Local Search: Two Neighbor Solutions

5 RESOURCE PROFILING

In the initial planning algorithm, the SSOWs were generated independently for each BSOP, using the SOWs that overlap the BSOP time range. The only exception was when a conflict had occurred between the OSSs of two BSOPs, then a complete SOW was explicitly forbidden, thus removed from the list used for SSOW generation. As OSSs from different BSOPs might already have booked the stations, this often resulted in conflicts or in solutions with bad quality.
Before generating the SSOWs for a BSOP, it was necessary to remove all the parts of the SOWs for which it was possible to infer that the corresponding station or satellite was already used, and then to generate the SSOWs using those ”filtered” SOWs.
There are two kinds of constraints that are not directly taken into account during the SSOW generation:

- the distance constraints with the previous and the next BSOP of the same User Service;
- the exclusive resource usage constraints.

The idea of the SOW filtering is to infer information from those constraints, by propagation. Currently the SOW filtering only takes the resource constraints into account.
The resource constraints are modeled in EPS as disjunctions of binary temporal constraints, linking the start and the end times of every two OSSs which should not overlap although their SOWs overlap. All the other constraints are either binary or linear. By propagating all the binary constraints, it is possible to infer the tightest bounds for the start and the end of each OSS, whatever the relative ordering of OSSs that must not overlap may be. From those bounds, in the case when the latest start time of an OSS is before its earliest end time, one can deduct that the OSS necessary takes place at least between those two time points. This means that the resources that the OSS needs, i.e. a station or a satellite, are necessary unavailable for any other OSS in this time range. Consequently, before generating the SSOWs for a BSOP, this time range can be subtracted from all the SOWs sharing at least one of the resources. One more important feature of the resource profiling is that it takes priorities between space crafts and ground stations into account. Imagine, that an OSS for space craft A had been implemented on a ground station G, and the priority for A on G is p_1. If during the next planning session another space craft, say B, is entitled to the same ground station with a higher priority p_2, so that the SOW generated for the B on G overlaps the $A's$ OSS on G, then it is desirable to create a conflict on G in order to generate a new SSOW with higher priority than before. More details on resource profiling can be found in [3].

6 CONSISTECY VALIDATION

As we already mentioned, in order to check the plan consistency, temporal relations between plan elements derived from the dynamic input to the EPS

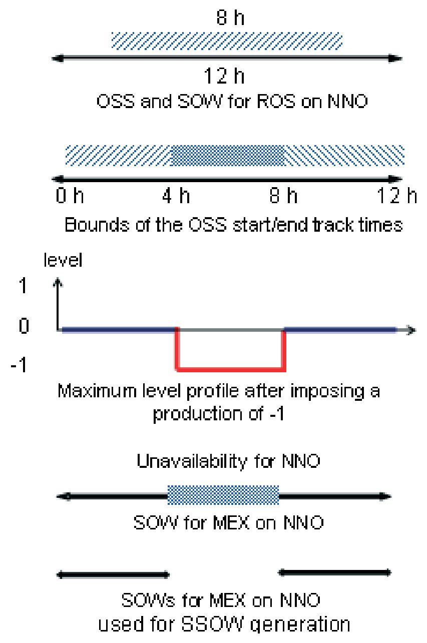
8 h
12 h
OSS and SOW for ROS on NNO
0 h
4 h
8 h
12 h
Bounds of the OSS start/end track times
level
1
0
-1
Maximum level profile after imposing a production of -1
Unavailability for NNO
SOW for MEX on NNO
SOWs for MEX on NNO used for SSOW generation

Figure 8: Resource Profiling

and its configuration will be expressed as a constraint network which will be examined on having a feasible solution. In such a way, the generation of a feasible contact profile will be turned into the *Constraint Satisfaction Problem* (CSP), which contains binary constraints, linear constraints and disjunctions of binary constraints.

- Linear constraints are the algebraic expressions of the form

  (10) $$\sum_i a_i t_i \leq b.$$

  They are widely studied in Linear Programming [4]. In the EPS context, linear constraints are used to restrict the overall duration of several events, e.g. $A_{end} - A_{start} + B_{end} - B_{start} \leq d$, or to enforce a particular ratio relation between two events, e.g. $A_{end} - A_{start} \leq r\,(B_{end} - B_{start})$.

- Binary constraints are linear constraints of the form

  (11) $$t_i - t_j \leq b,$$

  where t_i and t_j are the variables and b is a constant. Binary constraints are widely studied in *Simple Temporal Problems* (STP) [5]. The consistency check of the associated network is a cubic function of the number of variables. This class of constraints is used to enforce a particular order between events or to restrict a time gap between them. For instance if an event A has to precede some other event B then this relation can be expressed as $A_{end} - B_{start} \leq 0$.

- Disjunctive binary constraints have the following form

  (12) $$t_{i_1} - t_{j_1} \leq b_1 \vee \ldots \vee t_{i_n} - t_{j_n} \leq b_n.$$

  This type of constraints is widely studied in the context of *Disjunctive Temporal Problems* (DTP) [6]. These problems are NP-complete. In EPS, disjunctive binary constraints are used to ensure exclusive ground station usage by satellites. For example, the following disjunctive constraint $B_{end} \leq A_{start} \vee A_{end} \leq B_{start}$ will prevent communication sessions A and B to happen simultaneously on the same ground station.

Given that binary constraints are the majority of the constraints, and that STPs are far easier to solve than LPs, a sensible approach is to solve the DTP part of our problem, and check the linear constraints with LP only if a successful leaf is reached. Thus, the consistency of the overall constraint network is checked in two steps. First the DTP part is solved using a conflict-directed backjumping tree search algorithm with no-good recording called Epilitis [6]. Epilitis checks the consistency of a meta Constraint Satisfaction Problem. The variables of this meta-CSP are the disjunctions, the domain of each variable is the associated set of disjuncts, and the constraints between the variables are implicit. Thus an assignment to some variables is consistent if and only if the associated simple problem, i.e. STP or LP, is consistent. The search for a solution consists in the exploration of a tree, each node representing a partial assignment of the meta-CSP. The Epilitis algorithm was adapted for decomposed networks in order to limit the constraint propagation within the consistency checking process. The remaining linear constraints are processed using the simplex algorithm, or more precisely using the Phase I of the simplex algorithm, which gives us a feasible but in general not optimal solution.
In case of failure, we need to pinpoint the set of culprit constraints in order to derive the incriminated COSSes, thus to identify the incriminated BSOPs. Conflict directed strategies are clearly well suited to this as they use discovered conflicts to guide the search. Note that the meta-CSPs in the EPS context are dynamic CSPs. Each time a new BSOP is planned and COSSes are generated (resp. a COSS is removed consequently to a repair action), new time variables may be added (resp. removed), thus modifying the implicit constraints of the meta-CSP.

7 PLAN REPAIR

As already mentioned, COSSes are generated without any guarantee that the former underlying constraint network augmented with the new variables and constraints is consistent. If it is not the case, the incriminated COSSes must be detected and a repair action (to modify the COSSes from one BSOP) chosen.
When the meta-CSP is proven to be inconsistent, then the aim for a repair is to identify at least one *Minimal Unsatisfiable Subset* (MUS) [7] of the temporal constraints. A MUS is a set of conflicting constraints such that as soon as one of these constraints is removed, the resulting set is no longer conflicting. In our case, removing a COSS whose start or end time is involved in a MUS enables to solve the conflict identified by this MUS. See [8] and [9] for algorithms to

generate MUSes for DTPs and LPs.
Among the COSSes identified in a MUS, one must be removed. This choice takes into account general preferences such as mission to ground station priorities in case of a conflict on a resource, and heuristics favoring the stability of the network in order to avoid endless repairs.
The repair process mentioned above is local, thus it is not guaranteed to end with a solution. To prevent an endless repair loop, a termination criterion is provided, such as a maximum number of repairs, or a maximum time spent in repair. If this limit is reached, the system reports a failure to the EPS operators together with a set of User Services the degradation of which should allow solving the extracted conflicts.

8 CONSTRAINT NETWORK DECOMPOSITION

Having the well studied and widely used CSP decomposition methods [10, 5], we still decided to develop another one decomposition approach. Why complicate things? There are two main reasons for this. The first one is the structure of the problems we solve. As one can see on the Figure 9 (other missions or their combinations have similar kind of structure) our problems can be often represented as a set of loosely connected cliques of highly connected nodes, which corresponds to the structure of the graphs representing real-world problems. Therefore, trying to find a cycle cutset will usually amount to nothing. Collapsing nodes in order to get a tree-shaped CSP will create a tree with just a few nodes comprising a lot of nodes from the initial CSP, which leads to an intolerable increase of number of constraints.
The second reason is that our aim is to reduce the constraints propagation run time and not to achieve a backtrack-free search on a CSP problem. That's why we are interested in the first place in a decomposition on the STP level.
Nodes of a meta-CSP graph represent DTP's meta variables whose values are the constraints. Hence, two nodes of a meta-CSP graph are linked by an edge if and only if they share a time point, e.g. nodes representing meta variables $N_1 : x - y \leq a$ and $N_2 : x - z \leq b$ will be linked. The meta-CSP graphs of the planning problems we deal with have very high vertex connectivity, which is caused by the BSOP-based constraint model, where a lot of time points have to be constrained relatively to the start time point of the planning range. Removing the start point on the STP level amounts to the STP decomposition and at the same time to the meta-CSP graph decomposition. As we already mentioned, a set of constraints will be inferred for each BSOP during the planning process. The start and end of each BSOP can be represented as a time point arising as a result of summing up the start time of the planning range and a particular offset. Thus, constraints ensuring that each COSS lies inside of a BSOP or a SOW always involve the time point corresponding to the start of the planning range. This constraint model implies that in the distance graph of the constraint network all time points are connected to the start point. Due to the context we work in, it is reasonable to make the assumption that the time points of remotely located BSOPs are just rarely linked by a constraint directly. Hence, the only time point keeping a distance graph connected is the start time point, whose removal will break the distance graph into several parts having no time point in common. Since we cannot completely take out of consideration the start time point, we simply consider the *STPs having only this time point in common as disjoint STPs*. Figures 9 and 10 depict the decomposition effect on the constraint network of the Cluster planning session for one week time rage.

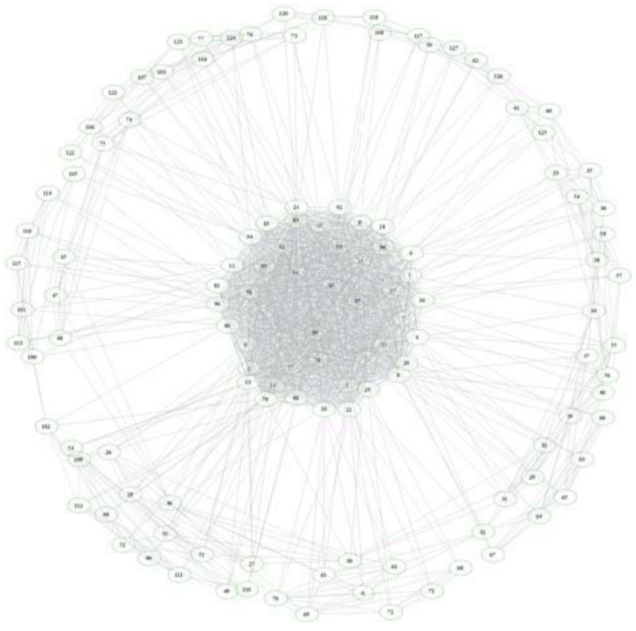
Figure 9: Meta-CSP Graph

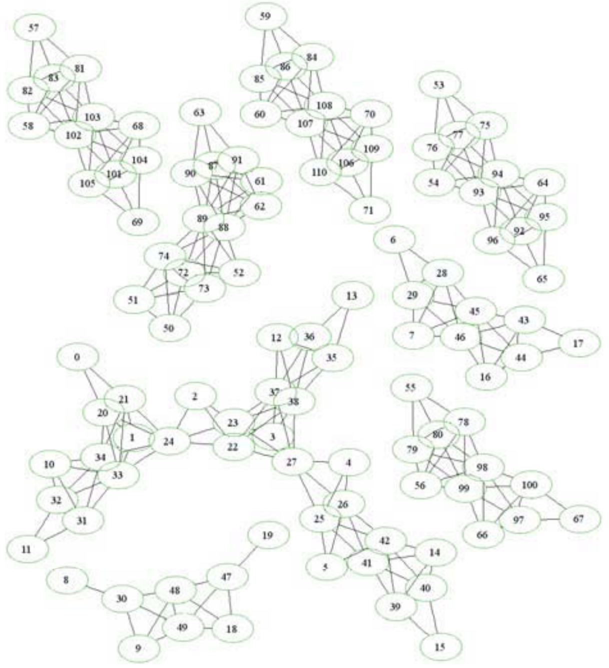
Figure 10: Decomposed Meta-CSP Graph

Having several STPs instead of a single one amounts into different approaches of constraint propagation, forward checking and meta variable subsuming. Let $C : x - y \leq a$ be a constraint which is about to be propagated. Three different cases can be now distinguished:

- x and y belong to the same STP;
- both or either time points are not yet in the constraint network;
- x and y belong to different STPs.

The propagation strategy in the first two cases is the same as it was without decomposition, i.e. one of the path consistency algorithms can be used [5]. The latter case has to be handled differently. Assume that distance graphs, and so STPs, are represented by matrices, where the item in the i^{th} row and j^{th} column denotes the distance between i^{th} and j^{th} time points.

Propagating a constraint, whose time points belong to disjoint STPs, can be performed in the following way: first, put the given STPs together, and then propagate the constraint as usual. Assume, STPs $A\,(n \times n)$ and $B\,(m \times m)$ don't have any time point in common, then the operation of putting them together amounts into trivial copying of their entries into a new matrix, $C\,((m+n) \times (m+n))$ and filling the corresponding $(m+n) \times (m+n) - m \times m - n \times n$ entries with ∞ indicating that no arc exists between these nodes.

If the only node connecting the given two STPs is the start node, then the entries of the resulting matrix depending on this node have to be filled properly. We illustrate the process of merging of two STPs in the Figure 11. Suppose that the start time point has in-

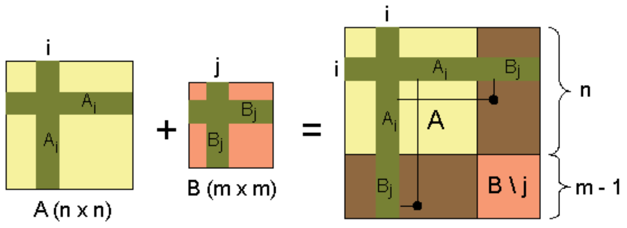

Figure 11: Matrix Merging Operation

dex i in the matrix A and in index j the matrix B, then entries of the resulting matrix which lie in the dark-brown-colored areas have to be composed as following:

$$c_{k,l} = \begin{cases} a_{i,l} + b_{p,j}, \; l \neq i & \text{if } k > n, \; l < n \\ a_{k,i} + b_{j,p}, \; k \neq i & \text{if } k < n, \; l > n \end{cases}, \text{ where}$$

$$p = \begin{cases} k - n & \text{if } k - n < j \\ k - n + 1 & \text{if } k - n > j \end{cases}$$

The reason for this is that the start node, s, connects nodes x_1 and x_2 from different STPs, and so the distance between x_1 and x_2 in the resulting matrix is the sum of distances between s and x_1, and s and x_2. Other parts of the resulting matrix can be simply copied from matrices A and B as shown on the Figure 11. The merging operation performs $n + m + 2\,(m-1)\,n$ arithmetical operations, assuming that A is an $n \times n$ matrix and B has dimension $m \times m$. Therefore, the complexity of the merging operation is $O\,(mn)$. After the matrices A and B have been merged, the constraint, which time points belong to A and B, can be propagated as usual. Since the disjuncts of constraints we deal with involve at most two time points, no more than two STPs will be merged in one Epilitis call. Note that the distance between two time points from different STPs can be now found by adding the distances between these time points and the start time point, which is available in every disjoint STP [11].

Operational tests showed significant improvements in the average run time of constraint propagation and consistency checking for the Epilitis version working on the decomposed STP. Moreover, the version using decomposition consumes much less memory than the one without decomposition. The figures below represent the run time and memory consumption comparison between Epilitis with and without STP decomposition for a planning session of the ENVISAT mission. The green graphs correspond to the results for the version with decomposition, the red ones to the results for the version without decomposition. Horizontal axis in each of the three cases represent the size of the given constraint network. The vertical axis in the first case represents the amount of memory used in kilobytes, and in the second and third cases it represents the amount of time in seconds spent on consistency checking or constraint propagation respectively.

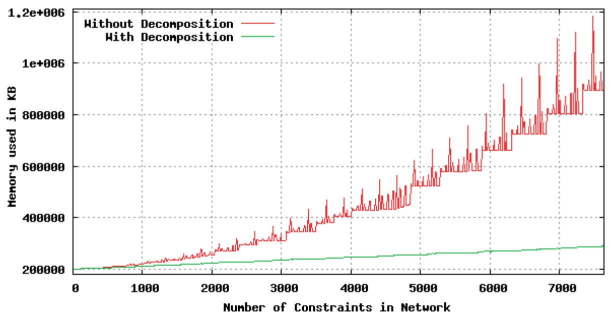

Figure 12: Memory Consumption Comparison

Note that in order to enable the backjumping, every recursive call of Epilitis has to store the STP it works with. It means that even if a small part of an STP has been changed, and the rest remains the same as in the previous Epilitis call, the whole STP has to be stored, which is a reason for the high memory consumption of the Epilitis version without decomposition. A significant improvement in this sense was achieved by the introduced STP decomposition. The version with decomposition maintains an array of disjoint STPs and stores only those STPs which have been changed in the current Epilitis call. Thus, the STP decomposition enables to avoid storing redundant information saving in this way a lot of memory.

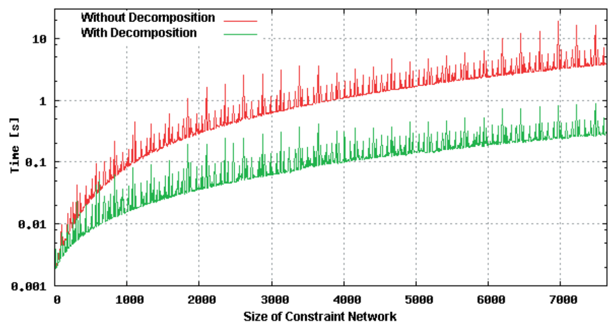

Figure 13: Consistency Checking Time Comparison

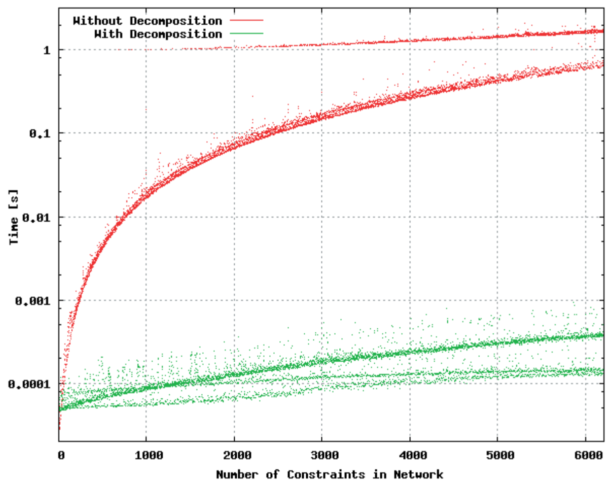

Figure 14: Propagation Time Comparison

9 FUTURE WORK

In addition to the already existent local optimization capability of the EPS, a **customizable objective function** allowing operators to choose between several local optimization criteria is currently under development. Another research area concerning the plan optimization in the EPS context deals with deployment of the **dynamic simplex algorithm**. Currently, for each BSOP the underlying LP will be solved from scratch, which can be sometimes avoided by using the ”warm start” feature of simplex. Having a solution to an LP P_1, one can use it to solve another LP, say P_2, consisting of the same objective function and an extended set of constraints. A faster solution to P_2 can be obtained by starting with the basis in the optimal solution to the P_1. In our case the extended set of constraints corresponds to the constraint network of the extended set of BSOPs.
Sometimes, it is impossible to generate a valid plan meeting the given communication requirements. The only reason for this are strictly formulated constraints in the mission agreement. Therefore, allowing formulating requirements with different level of strictness, will improve chances on finding a consistent plan. Our approach is to implement the **constraint network relaxation** feature based on the concept of soft constraints, which can be removed if they cause a conflict.
In addition, the current EPS development phase involves implementation of the **medium term planning** feature, i.e. creating one year plans for up to ten space missions in less than 36 hours, and implementation of the **long term load analysis** feature, which allows creating resource profiles of the ground station load on a station and mission basis for time periods up to ten years. The new features will provide analysis of the ESTRACK utilization helping to estimate its general capability to support further missions over their entire life cycles.

References

[1] Jrg Noll and Robin Steel. Eklops: An adaptive approach to a mission planning system. In *Proceedings of IEEE Aerospace Conference*, 2005.

[2] John Bather. *Decision Theory: An Introduction to Dynamic Programming and Sequential Decisions*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[3] Nicola Muscettola. Incremental maximum flows for fast envelope computation. In *Proceedings of International Conference on Automated Planning and Scheduling*, pages 260–269, 2004.

[4] George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, 1962. ISBN: 978-0-691-05913-6.

[5] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003. ISBN 1-55860-890-7.

[6] Ioannis Tsamardinos and Marta E. Pollack. Efficient solution techniques for disjunctive temporal reasoning problems, 2002.

[7] M. Liffiton and K. Sakallah. On finding all minimally unsatisfiable subformulas. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-2005)*, pages 173–186, 2004.

[8] John W. Chinneck and Erik W. Dravnieks. Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing*, 3(2):157–168, 1991.

[9] M. Liffiton, M. Moffitt, M. Pollack, and K. Sakallah. Identifying conflicts in overconstrained temporal problems. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 2005.

[10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2 edition, 2002. ISBN: 0-137-90395-2.

[11] Alexander Hoffmann. Solving dynamic scheduling problems with unary resources. Master's thesis, Darmstadt University of Technology, 2008.