# A Constraint-based Optimizer for Scheduling Solar Array Operations on the International Space Station

**Paper #700**


## Abstract

Space applications are demanding for complex operation and safety constraints and contain many objectives. This demand is even stronger for human spaceflight missions such as the International Space Station (ISS). This paper describes a novel approach to plan and schedule solar array operations on the ISS in a safe and effective manner. Opposite to previous approaches, it assumes global optimization while still taking in account safety and operation constraints. The paper proposes a constraint model to describe the problem formally and discusses methods to solve the model.


## 1 Introduction

Automated planning and scheduling deals with the problem of deciding which activities are necessary to reach the goal (planning) and when and where these activities should be executed (scheduling). These two tasks can be decoupled but if planning and scheduling are closely interconnected then it is more suitable to solve both tasks together. In this paper we present an integrated constraint-based planner and scheduler to generate schedules for solar array operations on the International Space Station (ISS).

Solar arrays at the ISS are designed to automatically track the sun, which is what they do most of the time. However, some ISS operations such as docking a spacecraft, extra vehicular activities, water dumps, thruster firings etc. impose additional constraints on solar array operations to prevent thermal stresses, environmental contamination, and structural loads. For such situations, solar arrays may need to be parked or even locked/latched, which must be planned in advance according to expected operations of the ISS.

Currently, the solar arrays planning problem is solved manually by a team of people known as PHALCONs (Power, Heating, and Lighting Controllers). It takes about four weeks to manually produce an ISS solar array operations plan for a typical four-week planning horizon. The Solar Array Constraint Engine (SACE) was proposed to automatically generate solar array operations plans subject to all operation constraints and user-configurable solution preferences [Reddy *et al.,* 2011]. The SACE uses an approach similar to manual scheduling with left-to-right greedy scheduling. The advantage is tractability of sub-problems solved, but the schedule is suboptimal and may not be found at all even if a feasible plan exists.

In this paper we propose a constraint-based approach that does global optimization. In particular, we formulate the problem as a constraint satisfaction problem where the planning component is modeled using optional activities with possibly zero durations. The problem formulation is taken mainly from the challenge domain at the International Competition on Knowledge Engineering for Planning and Scheduling 2012 [Frank, 2012]. Our approach and the SACE are the only two automated planners for this domain.

We first introduce the solar array operations planning domain and highlight the properties of the SACE approach. Then we describe our method in detail and discuss suggested solving techniques for the constraint model. Finally, we experimentally evaluate the proposed model and compare it with the simulation of the SACE.

## 2 The Problem and Existing Approaches

This section sketches the main parts of the problem solved; full details are in [Frank, 2012] and in [Reddy *et al.*, 2011].

The ISS has eight solar arrays, each of which is mounted on a rotary joint called the Beta Gimbal Assembly (BGA). The solar arrays are split into two groups each of which consists of four solar arrays mounted via the Solar Array Rotary Joint (SARJ) to the station (Figure 1). Thus each panel has two degrees of rotational freedom, though one degree of freedom is shared between the panels in the same group. Each rotary joint can be in exactly one mode: Autotrack, Park, or Lock (Latch for BGA), or the joint can be turning between the modes. The state of each joint is also described by the angle of orientation (360 positions).

In the Autotrack mode, the onboard software automatically rotates the panel so its surface is pointing directly onto the sun to maximize energy generated. In the formal model a known constant speed of rotation is assumed for this mode. The Autotrack mode must last at least 90 minutes. In the Park mode, a drive motor is engaged to maintain the current array angle, while in the Lock and Latch modes, a physical barrier is engaged. Transition into and out of Lock/Latch modes takes 20 minutes.

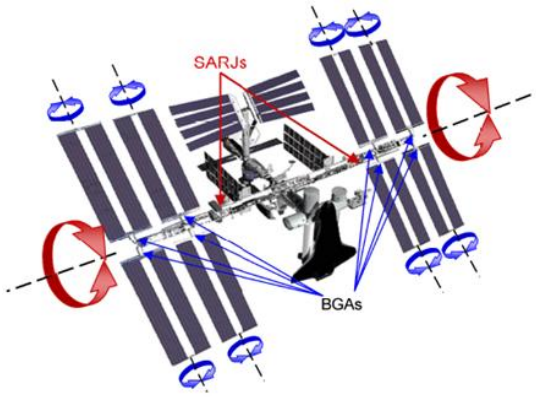


Figure 1: Solar arrays connected via rotary joints to ISS.

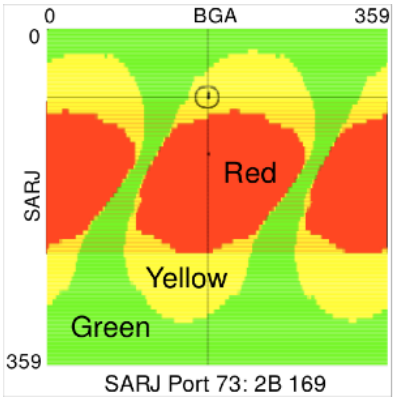


Figure 2: A table indicating, for one SARJ and one BGA, the safety areas for rotation angles.

The input to the solar array planning problem consists of a sequence of configurations, where each configuration starts at the time when the previous configuration finishes. This time may be flexible and in such a case the planner decides the appropriate time. This is the only situation when the groups of solar arrays interact; otherwise they can be scheduled independently. Each panel can be in exactly one mode in each configuration. The configuration also determines whether turning is disallowed (docking, undocking, reboost, maneuver), allowed at the end of the configuration (approach, prop purge, water dump), or allowed both at the beginning and at the end (attitude hold). It also defines a maximum rotation speed for SARJs (it is fixed for BGAs) and a contingency mode when some constraints can be violated. Finally, there are other parameters of the configuration determining for a pair of BGA and SARJ a set of four soft constraints: Power Generation (P), Structural Load (L), Environmental Contamination (E), and Longeron Shadowing (S). Each of these constraints is expressed as a $360 \times 360$ table (Figure 2) with three types of values: Green (preferred/best), Yellow (acceptable), and Red (infeasible in most situations/worst). The table is used as follows. If both BGA and SARJ are parked or locked/latched at some orientations then the value of the constraint is at the intersection of row and column corresponding to the orientations. If BGA (SARJ) is autotracking and SARJ (BGA) is parked or locked then the value of the constraint is the worst value in a row (column) defined by the orientation of SARJ (BGA). If both BGA and SARJ are autotracking then the value of the constraint is the worst value in the whole table. It is not allowed to use orientations with the red value for P, L, and S tables. Red value is allowed for the E table, but it is reflected in the quality of the plan. Turning of BGA can start only after turning of its SARJ finished and all BGAs (for a given SARJ) must start turning at the same time. The above tables are not assumed during turning.

The task is to decide for each joint in each configuration the following sequence of "activities":

- unlocking – transition out of lock/latch (optional)
- turning to the required orientation (optional)
- being in a selected mode
- turning to the next required orientation (optional)
- locking – transition into lock/latch (optional)

There might be some wait times between the above "activities", for example BGA turning waits until SARJ turning finishes. A joint can be in a selected mode in several consecutive configurations to satisfy the minimal duration constraint of the mode.

The plans are evaluated using four criteria. The most important criterion is color in the tables: table S first, then tables L and E, and finally table P (Figure 3). The second criterion is mode: Autotrack is preferred, followed by Park, and Lock/Latch to be last. Then, the number of changes in rotations should be minimized (the joint can rotate in positive and negative directions). Finally, the time spent in turning should be minimized. We are looking for a Pareto optimal schedule (no configuration can be scheduled better without worsening the schedule of another configuration).

## 2.1 SACE

The Solar Array Constraint Engine (SACE) is an automated planner for the solar array problem that mimics a human approach [Reddy *et al.*, 2011]. The SACE uses left-to-right scheduling. It decides the best orientation of joins and best modes in a given configuration (going from left to right). If this plan is not compatible with the plan for the previous configuration (not enough time for transition) then both configurations are merged and a new plan for the merged configuration is looked for. This also assumes "worst-cases" merge of the table constraints. Then look-ahead is used to prune infeasible modes for the next configuration. The process is repeated until a plan for all configurations is found (no backtracking to previous configurations is allowed).

The system is based on constraint solving, namely the SACE is build on top of the AI planning system EUROPA [Frank and Jónsson, 2003]. It exploits the tree structure of the constraint network to find a plan for a single configuration but due to the configuration merging, the schedule is suboptimal, and the algorithm could lead to a failure to find a valid plan even if one exists. This is the reason why we propose another constraint-based approach without these deficiencies.

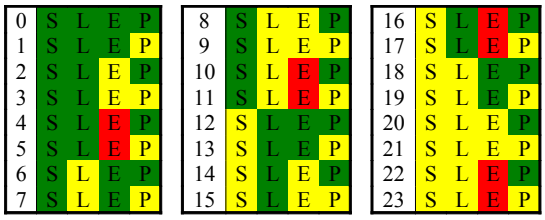


Figure 3: Color preferences order.

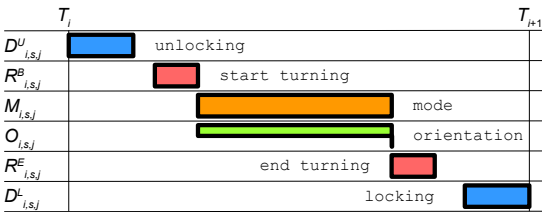


Figure 4: Time dependency of variables during a single configuration (ordering of activities within a configuration).

## 3 Constraint Model

This chapter describes the model used for solar array planing. First we define necessary variables and constants; then, we describe an objective function and, finally, we focus on different sets of constraints.

First, we describe the representation of the output plan. For each joint $j$ on the side $s$ in the configuration $i$ we introduce the following variables. $M_{i,s,j} \in \{AUTOTRACK, PARK, LOCK\}$ is a variable describing the mode of the joint. Each joint is in exactly one orientation (angle) during modes $PARK$ and $LOCK$ which is described by variable $O_{i,s,j} \in [0; 360)$ for the orientation of the joint. During the mode $AUTOTRACK$ the orientation changes continuously, so $O_{i,s,j}$ represents the orientation at the end of the mode before its final turning. Each joint can turn at the beginning or at the end of each configuration, so we have variables $R^B_{i,s,j}, R^E_{i,s,j} \in (-360; 360)$ for the angle and direction of the turning, where $R^B_{i,s,j}$ represents the turning at the beginning and $R^E_{i,s,j}$ represents the turning at the end of the configuration. Variable $T_i$ represents the start time of the configuration $i$ in minutes (also the end time of the configuration $i-1$).

The input defines the SARJ turn rate $S_i \in [9;\ 30]$ in degrees per minute and the type of the event (whether it is disallowed to turn joints, or to turn joints only at the end, or to turn joints also at the beginning of the configuration) $F_i \in \{NO; END; BOTH\}$ for each configuration $i$. Both $S_i$ and $F_i$ are constants. The input defines domains of variables $T_i$ (see above). The last part of the input is the set of color loss functions $c_{i,s,b}(M_{i,s,a}, M_{i,s,b}, O_{i,s,a}, O_{i,s,b}) \rightarrow [0;\ 23]$ that for each feasible combination of orientations and modes of BGA $b$ and corresponding SARJ $a$ on the side $s$ assigns a score of its color in the configuration $i$. This function combines tables P, L, E, S (Figure 2) with a given color preference order (Figure 3). It is represented as a table constraint defining the quality of schedule.

The following auxiliary variables are used only for the formulation of constraints. $D^U_{i,s,j}$ denotes how long the joint $j$ on the side $s$ must unlock at the beginning of the configuration $i$. Similarly, $D^L_{i,s,j}$ denotes the time of locking at the end of the configuration. Both unlocking and locking takes 20 minutes so domains of $D^U_{i,s,j}$ and $D^L_{i,s,j}$ are $[0; 20]$. Time dependency of variables is shown in Figure 4.

### 3.1 Objective function

There is no single global objective function given, there are only partial objective functions $f_{i,s}$, one for each configuration $i$ and side $s$. Each of these functions has four parts. The most important is the cost of colors:

$$f^C_{i,s} = \sum\nolimits_b c_{i,s,b}(M_{i,s,a}, M_{i,s,b}, O_{i,s,a}, O_{i,s,b}).$$

The second cost function is the cost of the modes:

$$f^M_{i,s} = \sum\nolimits_j m(M_{i,s,j}),$$

where $m$ is the loss function for modes meeting the condition $m(AUTOTRACK) < m(PARK) < m(LOCK)$.

The third cost function is the number of changes of directions $f^D_{i,s}$ that depends on $M_{i-1,s,j}$, $R^E_{i-1,s,j}$, $R^B_{i,s,j}$, $M_{i,s,j}$ and $R^E_{i,s,j}$. There are three positions, where the change of the direction may occur:

- before the $R^B_{i,s,j}$: $(R^E_{i-1,s,j} > 0 \wedge R^B_{i,s,j} < 0)$, or $(R^E_{i-1,s,j} < 0 \wedge R^B_{i,s,j} > 0)$, or $(M_{i-1,s,j} = AUTOTRACK \wedge R^E_{i-1,s,j} = 0 \wedge R^B_{i,s,j} < 0)$;
- before the $M_{i,s,j}$: $(R^B_{i,s,j} < 0 \wedge M_{i,s,j} = AUTOTRACK)$ or $(R^E_{i-1,s,j} < 0 \wedge R^B_{i,s,j} = 0 \wedge M_{i,s,j} = AUTOTRACK)$;
- between $M_{i,s,j}$ and $R^E_{i,s,j}$: $(M_{i,s,j} = AUTOTRACK \wedge R^E_{i,s,j} < 0)$.

The least important function is the cost of the turning length:

$$f^L_{i,s} = \sum\nolimits_j |R^B_{i,s,j}| + |R^E_{i,s,j}|.$$

The optimal plan of the configuration $i$ at the side $s$ is the plan with the lexicographically smallest vector $(f^C_{i,s}, f^M_{i,s}, f^D_{i,s}, f^L_{i,s})$.

### 3.2 Locking & unlocking

No joint can be simultaneously locked and unlocked, but the unlocking time or the locking time can be longer than the duration of the configuration. This restriction is defined using the constraint:

$$D^U_{i,s,j} + D^L_{i,s,j} \leq T_{i+1} - T_i \vee D^U_{i,s,j} = 0 \vee D^L_{i,s,j} = 0.$$

If some joint is locked during the configuration *i* and if it isn't locked in the configuration $i+1$, then it must be unlocked during the first 20 minutes of the configuration $i+1$. If the configuration $i$ is too short for unlocking, the remaining unlocking time is propagated to the configuration $i+1$. These restrictions are represented by the set of constraints:

$$M_{i+1,s,j} = LOCK \Rightarrow D^U_{i+1,s,j} = 0,$$

$$(M_{i,s,j} = LOCK \wedge M_{i+1,s,j} \neq LOCK) \Rightarrow D^U_{i+1,s,j} = 20,$$
$$(M_{i,s,j} \neq LOCK \wedge M_{i+1,s,j} \neq LOCK) \Rightarrow$$
$$D^U_{i+1,s,j} = max\{D^U_{i,s,j} - T_{i+1} + T_i; 0\}.$$

The situation is symmetric in the case of locking and similar set of constraints is used.

### 3.3 Turning & Autotrack

The type of the event must allow turning, which is modeled as:

$$F_i = END : R^B_{i,s,j} = 0,$$
$$F_i = NO : R^B_{i,s,j} = 0 \wedge R^E_{i,s,j} = 0.$$

The joint cannot turn, if it is locked, i.e.

$$M_{i,s,j} = LOCK \Rightarrow R^B_{i,s,j} = 0 \wedge R^E_{i,s,j} = 0.$$

There must be enough time to execute turning. It is a complex rule, so we only sketch the principle of the constraint, which is implemented using the *case* constraint in SICStus Prolog [Carlsson *et al.*, 1997]. For the beginning of the configuration: first, we compute when SARJ finishes turning, because turning of BGA can start only after turning of its SARJ finished; then we compute when the last corresponding BGA can begin turning, because all BGAs (for a given SARJ) must start turning at the same time. It is also necessary to take into account possible unlocking of joints. For the end of the configuration: first, we compute when all BGAs must start their turnings; then we can compute when SARJ must finish its turning. Finally, it is necessary to check whether there is enough time for turning on each joint.

Similar calculation is used for the duration of the Autotrack (the Autotrack mode must last at least 90 minutes). This constraint is even more complex, because the elapsed time may propagate to the next configuration if the configuration is too short, similarly to the case of unlocking/locking. The time spent in the Autotrack mode is further used for calculation of the final orientation, because in the case of the Autotrack mode, the orientation changes continuously. Full description of these constraints can be found in [Anonymous, 2014].

## 4 Optimization algorithm

The plan for each configuration $i$ and side $s$ has assigned a vector of four values $(f^C_{i,s}, f^M_{i,s}, f^D_{i,s}, f^L_{i,s})$ that evaluates its quality (lexicographically). To evaluate the quality of the whole plan we need to put together all these quadruples into a single vector while respecting the priorities within each quadruple. One option is to simply concatenate these quadruples in the order of configurations, but this would prioritize the first configuration "too much" ($f^L_{i,s}$ would be more important than $f^C_{i+1,s}$). Hence we decided to construct the global evaluation vector by putting first the values $f^C_{i,s}$ for all configurations and sides, then the values $f^M_{i,s}$ followed by $f^D_{i,s}$ and finally $f^L_{i,s}$. We have chosen this option, because the achievable color depends on the length of the turning nonlinearly and because a small change in the length of the turning can result in a significant difference in the achievable color. This approach should result in less stress on solar panels.

Now we describe, how the multi-criteria optimization is realized to get the lexicographically best solution. First we find an optimal solution (assignment of all variables) for the first objective function. Then we solve the problem again, but with an extra constraint binding the value of the first objective function to the just computed optimum and optimizing the second objective function etc. until all functions are optimized. This way we find the lexicographically best solution. So for every extended problem at this level of abstraction there exists a solution, if the first search was successful. Consequently it is possible to interrupt the algorithm at any time and get at least partially optimized plan, if it isn't enough time for complete optimization.

### 4.1 Optimization of a single objective

Branch and bound method [Land and Doig, 1960] is used to find an optimal solution for a given objective function. First, any solution is found. Then the algorithm is looking for another solution that is better than the last found solution until this extended problem has no solution or until the lower bound of that function is reached. It is possible to use suboptimal solutions if finding the optimal solution (or proving its optimality) is too computationally expensive. Therefore all iterations are running with a time limit.

When searching for the solution, classical methods for solving CSPs are used, that is, labeling of variables via depth-first search with maintaining consistency. Various variable ordering heuristics and branching schemes can be used during labeling. Different strategies can have different success rates and runtimes in different situations. Therefore we don't use one fixed strategy, but we exploit a set of strategies.

During optimization of each objective function (except the first one) the value of the current objective function from the last solution found during optimization of the previous objective function is used as the first upper bound of the current objective function. That is possible because no unfeasible constraints have been added – in the worst case we prove optimality of the previous solution according to the current objective function

When an objective function is optimized for the first time, no search is necessary if its upper bound is equal to its lower bound. This saves time, because the evaluation of the function on the known plan is faster than finding the whole plan. The method for determining the lower bound depends on the type of the optimized function.

- Optimization of $f^C_{i,s}$: 0 is the lower bound for the first iteration. It is because in our test cases in most cases there exists an orientation with all color values equal to 0 (see Figure 3). If the first iteration is stopped by timer (i.e. optimality of the solution isn't proved), we create a new problem which is composed only of the configuration $i$ and the side $s$. Then we find an optimal value of $f^C_{i,s}$ for this small problem. The optimal value from this small problem defines a new lower bound in the full problem (computation of the optimal value of the plan for one configuration is significantly easier than computing optimal values in larger plans).
- Optimization of $f^M_{i,s}$: Lower bound is computed before

the first optimization. The method of the computation is similar to the previous case. The only difference is that the fixed value of corresponding $f^C_{i,s}$ should be taken into account during optimization of the small problem, because $f^C_{i,s}$ is optimized prior to $f^M_{i,s}$ and because the constraint on the value of $f^C_{i,s}$ doesn't allow the Autotrack mode unless all tables are monochromatic.

- Optimization of $f^D_{i,s}$ and optimization of $f^L_{i,s}$: 0 is set as the lower bound, because the complete calculation of the lower bound is too time-consuming.

### 4.2 Simplified models

Solar array planning is a hard problem so we used simplified models which speed up the optimization without significant restriction of the solution space. The simplification of the model is obtained by additional constraints, so every solution in this simplified model is also a solution of the original problem. However, the optimal solution in the simplified model may not be optimal in the original model. So the optimal solution from the simplified model must be validated, i.e., the solution quality serves as the upper bound in the full model. This validation has higher success rate than search from scratch in the full model. We used the following two simplified models.

Our experiments showed that the Autotrack mode has the biggest influence on the difficulty of the problem. It is because this mode causes the orientation at the beginning of the configuration to be different from the orientation at the end and these differences are much harder to predict than differences caused by turnings. Functions $f^C_{i,s}$ are optimized first and they don't take modes into account, so we can omit the Autotrack mode during optimization of $f^C_{i,s}$. It cannot be omitted later, because the Autotrack mode is the best rated mode during optimization of $f^M_{i,s}$ and because functions $f^M_{i,s}$ are optimized prior to $f^D_{i,s}$ and $f^L_{i,s}$. This omission can be done because in most cases the result of the Autotrack mode is the same as the result of the Park mode with turning, and because if turning is prohibited, there is a high chance that the Autotrack mode will be prohibited too.

During turning, each orientation (except the initial orientation) can by achieved in two ways: (a) turning in the positive direction by angle $\alpha$, and (b) turning in the negative direction by angle $360 - \alpha$. A simplified model can have each orientation accessible in only one way. It can be enforced in two ways:

- The negative direction is prohibited, i.e. $R^x_{i,s,j} \geq 0$. Symmetrically it would be possible to prohibit the positive direction, but the negative direction causes changes of directions in contrast with the positive direction.
- The angle of turnings is restricted to interval $[0; 180]$, i.e. $|R^x_{i,s,j}| \leq 180$. One position is still accessible in two ways – if the angle of turning is $180°$, it doesn't matter whether the direction is positive or negative. However, restriction of the angle to $[0; 180)$ in one direction showed to be less effective.

This simplification is used during optimization of the function $f^M_{i,s}$. We used the second approach, because it gives better results than the first one.

## 5 Experiments

### 5.1 Test Cases

There is a set of test cases attached to the definition of the problem [Frank, 2012], but its size and difficulty is insufficient for testing of the algorithm and parameter tuning. Due to confidentiality it was not possible to obtain real data, so we have generated a dataset by replacing tables of the most difficult original test case from [Frank, 2012] with newly generated tables. The original test case contains four configurations in total duration seven hours. We have created 100 test cases which are more difficult to solve than the original test cases. Tables were generated as follows:

- Start with a green table.
- Add a yellow region with probability $\frac{1}{4}$ to the table. The yellow region is a horizontal stripe, a vertical stripe, or a rectangle. Its dimensions are taken randomly from interval $[120; 300]$ with uniform probability.
- Add a red region with probability $\frac{2}{3}$ to the yellow region. The red region has the same shape as its parental yellow region and yellow margin is at least 30px wide.

The planning horizon is only seven hours long in contrast to the four-week real planning horizon. However, data collected from ISS Live![1] during one month showed that short problematic periods (a few hours) are interspersed with a several-days long periods when all joints are in the mode Autotrack. So in fact, it is necessary to generate plans only for short periods, because joints are in the mode Autotrack in the rest of the time. Figure 5 shows one case where some joints aren't in the mode Autotrack.

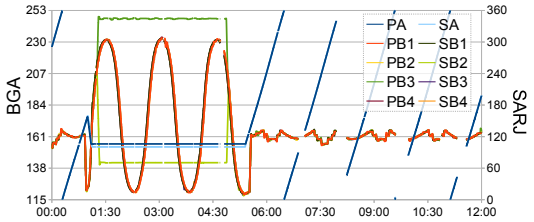


Figure 5: The example of real orientations of joints during half a day.

### 5.2 Parameter Tuning

Our algorithm contains two parameters that need to be tuned. The first one is the labeling strategy used in the search algorithm and the second one is the value of the time out.

We use SICStus Prolog and its CLPFD library [Carlsson *et al.*, 1997] for implementation of the solar array planing model, so we exploited its variable ordering heuristics and branching schemes too. It might be appropriate to develop heuristics specialized for the problem of solar array planning, but general heuristics were able to optimize all original test cases. We do not use all combinations of variable ordering

[1] http://spacestationlive.nasa.gov/

| *Method* | *Time [s]* | *Number of suboptimal plans* | | | | *Average value in final plans* | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $f^C_{i,s}$ | $f^M_{i,s}$ | $f^D_{i,s}$ | $f^L_{i,s}$ | $f^C_{i,s}$ | $f^M_{i,s}$ | $f^D_{i,s}$ | $f^L_{i,s}$ |
| **Final** | 1075 | 0 | 10 | 7 | 19 | 1.02 | 3.24 | 0.01 | 36.61 |
| **Wider** | 19758 | 0 | 2 | 8 | 14 | 1.02 | 3.19 | 0.01 | 32.19 |
| **Perpendicular** | 1369 | 8 | 5 | 1 | 7 | 2.15 | 3.21 | 0.01 | 21.57 |
| **Greedy** | 725 | 0 | 0 | 0 | 0 | 1.71 | 3.68 | 0.00 | 87.24 |

Table 1: Comparison of different approaches. Time is the total time in seconds required to evaluate 100 scenarios. Number of suboptimal plans indicates the number of plans out of 100 that the corresponding approach considers as suboptimal with respect to the corresponding part of the objective function, i.e. a greedy algorithm can pass a suboptimal solution off as the optimal solution, if this algorithm cut the branch with the optimal solution off at some stage of search. The average value in final plans shows the average value of the corresponding part of the objective function across all scenarios. Scores of colors $f^C_{i,s}$ are from domain [0; 92], scores of modes $f^M_{i,s}$ are from domain [0; 10], numbers of changes of direction $f^D_{i,s}$ are from domain [0; 15] and lengths of turning $f^L_{i,s}$ are from domain [0; 3590]. "Final" is the algorithm proposed in this paper. "Wider" is a variation of "Final" with wider portfolio of variable ordering heuristics and branching schemes. "Perpendicular" is the algorithm that optimizes objectives in the order $f_{0,s}, f_{1,s}, \ldots, f_{n-1,s}$. "Greedy" is our implementation of the SACE approach.

heuristics and branching schemes, because that is time consuming; we selected only a few pairs with the best results in initial experiments. We use two pairs during optimization of $f^C_{i,s}$ (namely (max,step) and (ff,bisect)) and $f^M_{i,s}$ (namely (ffc,bisect) and (leftmost,bisect)), one pair (min,bisect) during optimization of $f^D_{i,s}$, and two pairs (leftmost,bisect) and (min,bisect) during optimization of $f^L_{i,s}$.

As time outs were chosen 80ms for optimizing each of $f^C_{i,s}$ and $f^M_{i,s}$, 160ms for optimization of $f^D_{i,s}$, and 800ms for optimization of $f^L_{i,s}$. These values were chosen based on the experiments.

### 5.3 Results

We compare the proposed algorithm "Final" with some alternative approaches. The results are summarized in Table 1. All optimization algorithms were implemented in C# with .NET Framework 4 and they use the CLPFD library from SICStus Prolog for labeling of variables. All algorithms except of "Greedy" use the same time outs (see above), "Greedy" uses 20x longer time outs. Testing was performed on the laptop with an Intel® Core™ i5-520M processor (2.40GHz) and 8GB PC3-8500 DDR3 SDRAM.

First we compare the "Final" algorithm with a variation "Wider" that uses a full portfolio of variable ordering heuristics and branching schemes provided by the CLPFD library (five variable ordering heuristics and three branching schemes) to show that we have chosen an appropriate subset of available heuristics. The "Final" algorithm is only 1% worse in the score of $f^M_{i,s}$, so there is only little space for improvement. The difference in $f^L_{i,s}$ is more significant; the "Wider" is about 12% better, but the "Wider" is still not able to optimize 14 plans, i.e., it finds only five more optimal plans. Moreover, each expansion of the portfolio of heuristics of the "Final" will have a negative effect on the runtime.

"Perpendicular" is the algorithm that optimizes plans in configurations, i.e., all objective functions of one configuration are optimized before any objective function of another configuration is optimized – the order of objective functions is $(f^C_{0,s}, f^M_{0,s}, f^D_{0,s}, f^L_{0,s}, f^C_{1,s}, f^M_{1,s}, \ldots, f^D_{n-1,s}, f^L_{n-1,s})$ – so it is more similar to left-to-right scheduling of the SACE approach, but it isn't a greedy algorithm. It has about 41% better score than "Final" in the least important part of the objective function at the expense of the most important part of the objective function, where it has about 111% worse score. So these plans put more strain on solar arrays.

"Greedy" is our implementation of the SACE approach as we understood it from [Reddy *et al.*, 2011], because their implementation isn't publicly accessible. It needs only about 33% less time for optimization of all plans, but the average score is significantly worse in all parts of the objective function except $f^D_{i,s}$. The average score of $f^C_{i,s}$ is about 67% worse, the average score of $f^M_{i,s}$ is about 14% worse and the average score of $f^L_{i,s}$ is about 138% worse. So our approach is able to provide significantly better plans in a slightly longer time.

## 6 Concluding remarks

This paper shows that constraint-based technology can be applied to ISS solar array operations scheduling with highly complex transition constraints and objectives. We proposed a constraint model and a special layered optimization algorithm that exploits a portfolio of classical variable ordering heuristics and branching schemes. We experimentally showed that this approach generates much better schedules than the existing left-to-right scheduling while keeping similar time efficiency. Though this global optimization approach does not scale yet to a full four-week horizon, based on data collected from ISS, the periods, when something happens and scheduling is complex, are short enough to be covered by our optimizer. The major contribution is showing that off-the-shelf technology with some specialized optimization procedure overcomes an ad-hoc solver in terms of schedule quality while keeping time efficiency comparable. Full technical details are available in [Anonymous, 2014].

## References


[Anonymous, 2014]. Diploma Thesis at Some University, 2014

[Carlsson *et al.*, 1997] M. Carlsson, G. Ottosson, B. Carlsson. An Open-Ended Finite Domain Constraint Solver. In *Programming Languages: Implementations, Logics, and Programs*, 1997.

[Frank, 2012] J. Frank. Planning Solar Array Operations on the International Space Station. In *The International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS 2012)*, 2012.

[Frank and Jónsson 2003] J. Frank, A. Jónsson. Constraint-Based attribute and interval planning. *J. Constraints 8*, 4, 339–364, 2003.

[Land and Doig, 1960] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. In *Econometrica 28*, 3, 497–520, 1960.

[Reddy *et al.*, 2011] S. Reddy, J. Frank, M. Iatauro, M. Boyce, E. Kürklü, M. Ai-Chang, A. Jónsson. Planning Solar Array Operations for the International Space Station. In *ACM Transactions on Intelligent Systems and Technology* (TIST), Volume 2, Issue 4, 2011.