# Planning for Spatial Missions Using Answer Set Programming

Rodrigo Santos
Instituto Superior Técnico / INESC-ID
rodrigo.santos@ist.utl.pt

**Abstract**

In this work, we use the declarative language of Answer Set Programming to solve the crew planning problem: to find an optimal work schedule for each crew member of a spatial mission in the International Space Station, while respecting all the constraints associated to each task. We introduce the ASP language, the programming methodology and the tools we use. We present four dedicated programs to solve the described problem, elaborating on ASP program design and optimization of the grounding and solving processes. The encodings follow two separate models which are based on different interpretations of the problem's constraints. Finally, we compare our work to planners submitted to the International Planning Competition.

**Keywords**

Answer Set Programming, Artificial Intelligence Planning, Crew Planning, Scheduling

## 1 Introduction

The International Space Station (ISS) is a permanently inhabited station, that orbits around the Earth. It is a joint effort of five international partners: USA, Russia, Japan, Canada and Europe. These partners work together in order to develop their space programs, while studying the conditions of life in outer space. The low-gravity environment of the ISS allows the conduction of scientific experiments that would otherwise be impossible.

The planning of all ISS activities is greatly detailed. Each year of the mission goes through several planning stages: planning starts with a document describing the ground rules and constraints of the mission, a first plan is created that summarizes the priorities for the year, which gets more detailed in several planning iterations. The final product is a collection of short-term plans that are executed in the ISS. In this work we will address a small part of this planning structure, which concerns the short-term plans of the activities of the crew members.

Answer Set Programming (ASP) is an up-and-coming declarative language that has earned a place in Knowledge Representation and Artificial Intelligence conferences, despite its youth. Its appeal comes from being an expressive, efficient and easy to use language. Because of this, we adopted ASP as the technology used in this work. Our goal is as much to solve the planning problem of the ISS, as it is to explore the potential of ASP for the resolution of planning problems.

## 2 Crew Planning

The crew planning problem consists in designing a schedule for the activities of the crew members of the ISS, over a number of days[1]. The scheduled plan must be feasible – each crew member can perform at most one activity at any instant. If an activity is part of the plan, then all its constraints must be respected. There are also medical restrictions that cannot be violated; in order to ensure the safety of the crew members, and as such some activities are enforced in the plan.

The basic plan, with activities that are mandatory for every crew member to perform every day, is composed of the following activities:

- A **post-sleep period** of 195 minutes (3 hours and 15 minutes), at the beginning of the day.

- A **sleep period** of 600 minutes (10 hours), at the end of the day.
- An interval for **lunch** of 60 minutes (1 hour). The lunch period can be scheduled anywhere between the post-sleep period and the sleep period.
- An **exercise** period of 60 minutes. A period of exercise is associated with a machine, and each machine can be used by only one person at a time.

The tasks in the basic plan are the only mandatory tasks in a schedule. Every other task must be specified in the problem instance in order to be present in the schedule.

A **payload** task is a task with the duration of 60 minutes, and a deadline which limits the latest day at which the task can be performed. Payload tasks can be performed by any crew member, on any day up to the deadline.

A **medical conference** has the duration of 60 minutes. This task is performed by a specific crew member in a specific day. This information is stated in the problem instance.

A **filter change** task has the duration of 60 minutes. The task is scheduled for a specific day, but can be performed by any crew member.

The replacement of the **Remote Power Controller Module** (RPCM) is a complex task that consists in several smaller tasks, which are executed in the following order:

## RPCM sequence

1. - Reconfigure Thermal Loops
   - Remove Sleep Station
2. Replace RPCM
3. - Reconfigure Thermal Loops
   - Assemble Sleep Station

All tasks have the duration of 60 minutes, with the exception of *Replace RPCM* which has a duration of 180 minutes. Each task in the sequence must be finished before the next task in the sequence starts. Each task can be performed by any crew member independently. The whole sequence of tasks must be completed until an established deadline day.

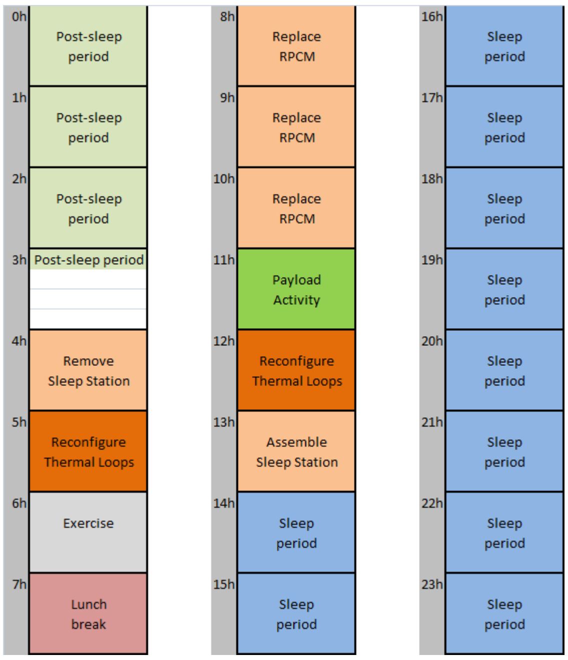

Figure 1: Schedule of one day

Figure 1 shows an example plan of one day for one crew member, which contains the mandatory activities from the basic plan, as well as one sequence of RPCM tasks and one payload task. The blank space in the figure represents time that is not allocated to any task.

The crew planning problem is similar to a **machine scheduling**[2] problem. In machine scheduling, we use the concepts of **job** and **machine**. A job is a set of related tasks, which may have a defined order of execution. A machine is a resource, which is used to perform a task. Generically, a machine scheduling problem consists in, given a number of jobs and machines, schedule the execution of the jobs, defining for each task which machine is going to perform the task and when. A machine cannot perform two tasks at the same time, and tasks of the same job cannot be performed at the same time.

## 3 Answer Set Programming

Answer Set Programming(ASP) [3, 4, 5] is a logic programming paradigm, which consists in computing sets of literals (**answer sets**) that solve a given logic program. ASP originated from the stable model semantics[6] and has close ties with non-monotonic reasoning and default logic. Programs in ASP are written in a truly declarative way, with a syntax similar to that of Prolog.

ASP uses the logic concepts of term, atom and literal. Rules are expressions of the form

$$l_0 \leftarrow l_1, ..., l_m, \text{not } l_{m+1}, ..., \text{not } l_n$$

Rules are implications – in the above rule, if each literal $l_1, ..., l_m$ is true and each literal $l_{m+1}, ..., l_n$ can be assumed to be false, then the literal $l_0$ is true.

The left side of the rule is called the **head**, and the right side of the rule is called the **body**. **Facts** are rules in which the body is empty, and represent literals that are always true. **Constraints** are rules in which the head is empty, and represent a conjunction of literals that cannot be true.

ASP also supports the use of **cardinality constraints** (also called *choice rules*), **weight constraints** and **conditional expressions**. These allow to choose a number of elements of a set to put in an answer, to assign weights to each element, and to constrain the value of a free variable. Further constraints can be expressed with the help of arithmetic functions and comparison predicates.

The process of obtaining the solutions from a program is divided in two stages: **grounding** and **solving**. In the grounding stage, the rules of a program are compiled by replacing all variables by their domain values. The result is a ground program. In the solving stage, the ground program is used to produce the answer sets. An answer set is a set of literals that satisfies all the rules in the ground program, and that can be derived from those rules.

There is not a universal methodology to solve problems using ASP. However, there are some techniques used by experts that can be taken as guidelines to help the programmer. One method in particular, **generate-and-test**[5, 7, 8, 9], is commonly used as a standard. It consists in dividing a program in three parts. First, the problem data is presented in the form of facts. Then, we write generator rules: choice rules that produce the predicates that will compose an answer set. Finally, we write constraints that eliminate invalid solutions. Once a basic and working encoding is established, one can delve further into optimizations, such as symmetry-breaking rules[10] and projection techniques[9].

**Gringo**[11, 12] and **clasp**[13, 12][1] are the state-of-the-art tools for grounding and solving in ASP, respectively, as well as the tools used in this work. Other ASP tools include **Smodels**[14][2], **DLV**[15][3], **ASSAT**[16][4] and **cmodels**[17][5]. A special mention goes to **iClingo**[18], a tool designed to solve iterative problems, and **ASPviz**[19][6], a tool that translates answer sets into a graphical representation.

## 4 Solution Development

We used two different formulations in this work: the "standard model" and the "competition model". They represent two different ways of approaching the crew planning problem. The standard model is a formulation that we believe represents the problem in a logical way and in a way that is adequate for the use of ASP. On the other hand, the competition model follows closely the domain model used in the **International Planning Competition**[7] (IPC).

The main difference of the competition model when compared to the standard model is that days do not have a fixed length. Instead, there is a minimum amount of hours required to have passed since the beginning of the schedule in order to advance to the next day, corresponding to 24 hours

[1]http://potassco.sourceforge.net/

[2]http://www.tcs.hut.fi/Software/smodels/

[3]http://www.dlvsystem.com/

[4]http://assat.cs.ust.hk/

[5]http://www.cs.utexas.edu/users/tag/cmodels/

[6]http://www.cs.bath.ac.uk/~occ/aspviz/

[7]The domain file can be downloaded in http://ipc.informatik.uni-freiburg.de/Domains

for each previous day (Day 2 cannot start after at least 24 hours, Day 3 after 48 hours, and so on). Also, crew members can work in different shifts. So the same day can have different lengths for different people. There are no constraints related to this, i.e. it is irrelevant to each crew member where the rest of the crew stands in their schedules.

We developed four different encodings during this work. The encodings follow the generate-and-test approach. The facts of our programs are the number of days, crew members, exercise equipments, and information about the tasks that change according to the instance and the encoding. However, all the tasks present in a plan must be described in the instance. A task always has an unique ID, a name and a type that identifies its duration. It also has information about its day of execution, or the deadline if the precise day is not determined. If the task is already assigned to a crew member or to a specific time slot, that information is also on the instance. RPCM tasks have additional information describing its place in a RPCM sequence of tasks. Instances of the competition model also have information about the time domain for that instance, which represents the maximum length of the plan. For the standard model, the time domain is fixed on 24 hours and the length of a plan varies only according to the number of days.

The resulting plan is obtained from a series of answer predicates (predicates that were generated by the solved with new information) that relate each task to each of the other existing domains. Answer predicates may give information about the day and time of a task, the crew member that performs it, or the exercise machine it uses (in the case of exercise tasks) and are obtained through generator rules.

Along with the generator rules, each encoding has constraints that prevent tasks from overlapping, either because they are performed by the same crew member or in the same exercise equipment at the same time. Additional constraints control the execution of RPCM sequences, so that the tasks in a sequence are performed in the correct order. The encodings that follow the compe-

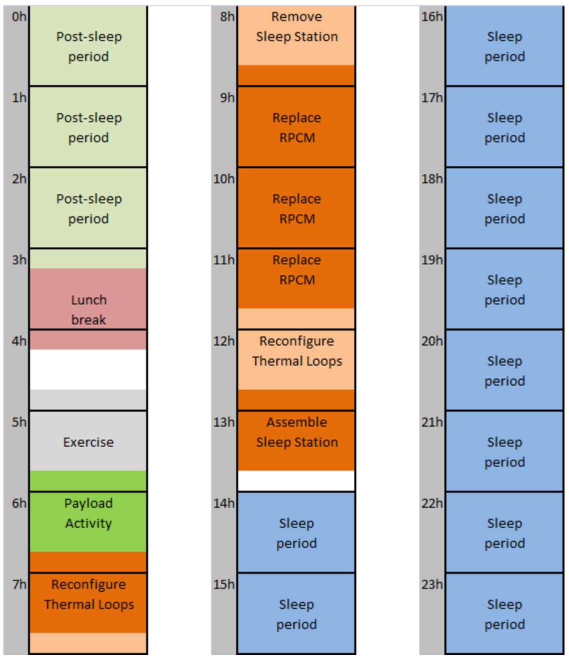

Figure 2: Schedule of one day using the standard model

tition model also have rules to manually limit the length of a day. In the standard model there are no such rules because the length is fixed.

For each model, there are two encodings: one basic encoding, which uses normal constraints and separate answer predicates for each kind of information, and one choice-rule based encoding, which uses constraints expressed as choice rules and agglomerates most information about a task in a single answer predicate.

The basic encodings were inefficient at start, so we optimized them by changing the rules that generate time predicates, in order to limit the domain of the time variable for specific tasks. In the standard model, the post-sleep and sleep tasks always happen at a fixed time, so we eliminated the timespan of those tasks from the domain of other tasks. In the competition model, we created several generator rules, that minimized the size of the time domain as much as possible, for each specific task. We also applied symmetry-breaking techniques into some of the program's constraints. As a result, the optimized versions have much smaller ground files.

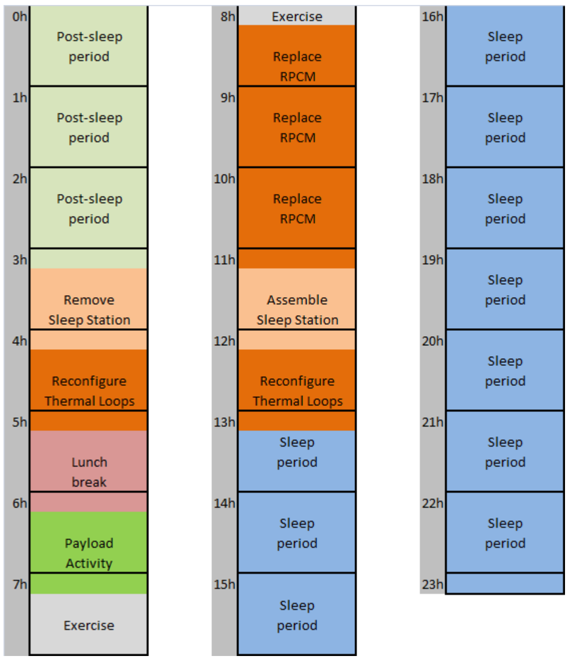

Figure 3: Schedule of one day using the competition model

## 5 Results

To evaluate this work, we used 30 instances obtained from the **International Planning Competition** (IPC)[8]. Each instance can require a plan that lasts from 1 to 3 days, and have 1 to 3 assigned crew members. The instances use the **PDDL**[20] language, so we went through a conversion step in order to translate the instances to ASP. The tests have an imposed limit of 30 minutes per instance and a memory limit of 3,8 GB. We used the program runsolver[21] to control the evaluation process.

The instances are compatible with the competition model. In their original form, some instances are impossible to solve in the standard model. In order to assure all instances were solvable, we trimmed the unsolvable instances by removing some tasks from them.

We started by comparing the performance of the ASP encodings. In this comparison, we ran gringo and clasp with the default options. Table 1 shows the number of solved instances for each encoding. Results are overall better for encodings of the standard model than for encodings of the competition model. The choice rule encoding for the standard model produced the most efficient ground files and with overall quicker times.

[8]The instances can be downloaded in http://ipc.informatik.uni-freiburg.de/Domains

Table 1: Solved instances by encoding

| Encoding | Solved |
|---|---|
| Standard Model: Basic | 18/30 |
| Standard Model: Basic Optimized | 22/30 |
| Standard Model: Choice Rules | 25/30 |
| Competition Model: Basic | 10/30 |
| Competition Model: Basic Optimized | 11/30 |
| Competition Model: Choice Rules | 13/30 |

Grounding performance is mostly regular. Although the growth in time and file size from smaller to larger instances is exponential, for the most part, the grounder spends more time when creating larger ground files. The solving performance is more extreme. Most of the time, either a program solves an instance very quickly (in under 5 seconds), or it does not solve the instance at all. Also, the ground files produced by big instances in competition model encoding are too large for the solver. In many occasions, clasp runs out of memory while performing the search, and in some cases the program cannot even finish reading the ground file.

We tested different configurations of clasp's parameters on the choice-rule based encodings. The manipulated parameters[12] in each case study allowed the pre-processing of choice rules, or enabled random probing. The best performance was obtained by using random probing with the choice-rule encoding of the standard model, which solved 28 of the 30 instances. We also concluded that the time spent on pre-processing increases the total time spent on the solving stage. We tested the **claspfolio**[22] tool with both encodings, but there was not an improvement in performance compared to the default configuration.

We also tested what were the implications of working with an suboptimal set of instances in the competition model, using the optimized encoding of the competition model. A suboptimal instance is an instance with a larger time domain than the required for an instance to be solvable. The ground files for suboptimal instances are larger than for optimal instances. However, the solver was more successful when solving suboptimal instances, despite the larger file size. It solved 18 of 30 suboptimal instances, compared to 12 of 30 optimal instances.

In a comparison to the best planners from the last two IPCs, **POPF2**[23, 24], from IPC-2011, solved all 30 instances, while **SGPlan6**[25], from IPC-2008 solved only 9 of the 30 instances optimally.

## 6 Conclusion

During the development of this work, our goal was set on solving the 30 instances used in the International Planning Competition. We also wanted to, as much as possible, present solutions similar to the ones given by the competition planners. However, there was a conflict of interest between what was the best approach to take in terms of maximizing ASP performance, and present a solution as similar as possible to the competition output. Because of this, our work diverged into two parts, which although very strongly linked, were fundamentally different.

We think ASP is stronger when dealing with binary problems, and thus it is not the more adequate approach to tackle planning problems. However, it still performed reasonably well, solving 28 of the 30 instances when using the best approach. This is proven by the differences between the standard model (which optimizes ASP performance) and the competition model (which attempts to simulate existing planners).

Some possibilities of future work involving ASP and the crew planning problem are to use a model with a different set of constraints in its tasks, solve a version of the problem based in maximizing the utility of the tasks included in a plan and to develop a generic planner/scheduler using ASP.

## References

[1] J. Barreiro, G. Jones, and S. Schaffer, “Peer-to-peer planning for space mission control,” in *Aerospace conference, 2009 IEEE*, pp. 1 – 9, march 2009.

[2] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, “Sequencing and scheduling: algorithms and complexity,” in *Logistics of Production and Inventory* (S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, eds.), vol. 4 of *Handbooks in Operations Research and Management Science*, ch. 9, pp. 445–522, Elsevier, 1993.

[3] V. W. Marek and M. Truszczynski, “Stable models and an alternative logic programming paradigm,” *CoRR*, vol. cs.LO/9809032, 1998.

[4] I. Niemela, “Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm,” *Annals of Mathematics and Artificial Intelligence*, vol. 25, no. 3, pp. 241–273, 1999.

[5] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press, Feb. 2003.

[6] M. Gelfond and V. Lifschitz, “The stable model semantics for logic programming,” in *ICLP/SLP*, pp. 1070–1080, 1988.

[7] M. Brain, O. Cliffe, and M. D. Vos, “A pragmatic programmer’s guide to answer set programming,” in *Software Engineering for Answer Set Programming (SEA09)*, pp. 49–63, September 2009.

[8] T. Syrjänen, “On the practical side of answer set programming,” in *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, pp. 473–489, 2011.

[9] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, “Challenges in answer set solving,” in *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, pp. 74–90, 2011.

[10] T. Mancini, D. Micaletto, F. Patrizi, and M. Cadoli, "Evaluating ASP and Commercial Solvers on the CSPLib," *Constraints*, vol. 13, no. 4, pp. 407–436, 2008.

[11] M. Gebser, T. Schaub, and S. Thiele, "Gringo : A new grounder for answer set programming," in *LPNMR*, pp. 266–271, 2007.

[12] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele, "A user's guide to gringo, clasp, clingo, and iclingo." Unpublished draft, 2010. Available at URL[9].

[13] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "*clasp* : A conflict-driven answer set solver," in *LPNMR*, pp. 260–265, 2007.

[14] P. Simons, I. Niemelä, and T. Soininen, "Extending and implementing the stable model semantics," *Artif. Intell.*, vol. 138, no. 1-2, pp. 181–234, 2002.

[15] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello, "The dlv system for knowledge representation and reasoning," *ACM Trans. Comput. Log.*, vol. 7, no. 3, pp. 499–562, 2006.

[16] F. Lin and Y. Zhao, "Assat: computing answer sets of a logic program by sat solvers," *Artif. Intell.*, vol. 157, no. 1-2, pp. 115–137, 2004.

[17] Y. Lierler and M. Maratea, "Cmodels-2: Sat-based answer set solver enhanced to non-tight programs," in *LPNMR*, pp. 346–350, 2004.

[18] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele, "Engineering an incremental asp solver," in *ICLP*, pp. 190–205, 2008.

[19] O. Cliffe, M. D. Vos, M. Brain, and J. A. Padget, "Aspviz: Declarative visualisation and animation using answer set programming," in *ICLP*, pp. 724–728, 2008.

[9] `http://heanet.dl.sourceforge.net/project/potassco/potassco_guide/2010-10-04/guide.pdf`

[20] M. Ghallab, C. K. Isi, S. Penberthy, D. E. Smith, Y. Sun, and D. Weld, "PDDL - The Planning Domain Definition Language," tech. rep., CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[21] O. Roussel, "Controlling a solver execution with the runsolver tool," *JSAT*, vol. 7, no. 4, pp. 139–144, 2011.

[22] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. T. Schneider, "Potassco: The potsdam answer set solving collection," *AI Commun.*, vol. 24, no. 2, pp. 107–124, 2011.

[23] A. J. Coles, A. Coles, M. Fox, and D. Long, "Forward-chaining partial-order planning," in *ICAPS*, pp. 42–49, 2010.

[24] A. Coles, A. J. Coles, A. Clark, and S. Gilmore, "Cost-sensitive concurrent planning under duration uncertainty for service-level agreements," in *ICAPS*, 2011.

[25] C. wei Hsu and B. W. Wah, "The sgplan planning system in ipc-6."