Metrics for Planetary Rover Planning & Scheduling Algorithms

J.M. Delfa Victoria*
Department of Computer Science
Technische Universität Darmstadt, Germany
delfa@sim.tu-darmstadt.de

N. Policella
ESA-ESOC
European Space Agency
Darmstadt, Germany
nicola.policella@esa.int

M. Gallant
ESA-ESOC
European Space Agency
Darmstadt, Germany
marc.gallant@esa.int

O. von Stryk
Department of Computer Science
Technische Universität Darmstadt, Germany
stryk@sim.tu-darmstadt.de

A. Donati
ESA-ESOC
European Space Agency
Darmstadt, Germany
alessandro.donati@esa.int

Y. Gao
Surrey Space Center
University of Surrey, UK
yang.gao@surrey.ac.uk

ABSTRACT

In addition to its utility in terrestrial-based applications, Automated Planning and Scheduling (P&S) has had a growing impact on space exploration. Such applications require an influx of new technologies to improve performance while not comprimising safety. As a result, a reliable method to rapidly assess the effectiveness of new P&S algorithms would be desirable to ensure the fulfillment of of all software requirements. This paper introduces *RoBen*, a mission-independent benchmarking tool that provides a standard framework for the evaluation and comparison of P&S algorithms. RoBen considers metrics derived from the model (the system on which the P&S algorithm will operate) as well as user input (e.g., desired problem complexity) to automatically generate relevant problems for quality assessment. A thorough description of the algorithms and metrics used in RoBen is provided, along with the preliminary test results of a P&S algorithm solving RoBen-generated problems.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

1. INTRODUCTION

Recently, robotics has been gaining prominence in several space scenarios such as planetary exploration, ISS exploitation and deep space missions. The complexity of these missions requires the infusion of new technologies in order to maximize performance while preserving the safety of the spacecraft. One of these technologies is Automated Planning and Scheduling (P&S), which gives the system the ability to make decisions about the actions to execute while considering its status and the changes in the environment. A number of missions from NASA and ESA have been equipped with different levels of on-board autonomy [12, 13, 6, 1] providing a great improvement in terms of cost savings, science return and safety, among other benefits [4, 7].

A relevant problem when introducing innovation is the assessment of the new technologies in order to provide adequate confidence to the customer that the software satisfies its requirements [9]. This is particularly important in the case of future missions where real test-cases may not be available to prove the adequacy of new solutions [14]. The development and introduction of these new concepts should be supported by ad-hoc methodologies and techniques to measure the final expected performance of the system.

In particular, this paper focuses on the assurance of software product quality, while program processes and implementation are not within the scope of this study. We have identified *Functionality* [8] as the characteristic to be measured, more precisely the *Efficiency* of the product. For this sub-characteristic, we have identified a number of metrics that can be classified in two different groups:

- Problem complexity metrics that analyse the complexity of the generated problem.
- Performance metrics that focus on the performance of the P&S algorithm.

It is worth remarking that problem complexity metrics are crucial to deeply understand the results obtained by using the performance metrics. In fact, good performance results are valuable only when obtained on (very) complex problem instances. Problem complexity metrics can be further divided between:

- Static metrics that only consider the problem input and initial

*Also affiliated to Surrey Space Center, University of Surrey, UK and to ESA-ESOC, European Space Agency, Darmstadt, Germany

PerMIS'12 Hyattsville, Maryland USA

values (e.g., resource availabilities) to calculate the complexity.

- Dynamic metrics that also consider the estimated behavior of the system during the plan execution.

For example, dynamic metrics can consider the resources available at the specific time each individual task should be executed, the initial amount, the expected consumption for each task already executed, and the expected amount of resources generated during execution.

This paper presents a new mission-independent benchmarking tool called RoBen, which is currently under development. The main objectives of RoBen are to provide a standard framework to evaluate and compare automated planning tools as well as to help future operators validate alternative plans. In particular, RoBen will automatically generate problems to evaluate the quality of P&S algorithms for rover scenarios.

A set of metrics assembled for use with RoBen is introduced in the paper. While the performance of software products has been widely studied, problem complexity in real scenarios such as space robotics remains quite immature and dependant on the specific characteristics of the mission and/or planning tools. Therefore, a combination of metrics both novel and borrowed from the literature [8, 11, 5] have been used to improve the results.

Finally, we present some preliminary results with different autonomously generated problems evaluated by a general purpose planner developed at ESA.

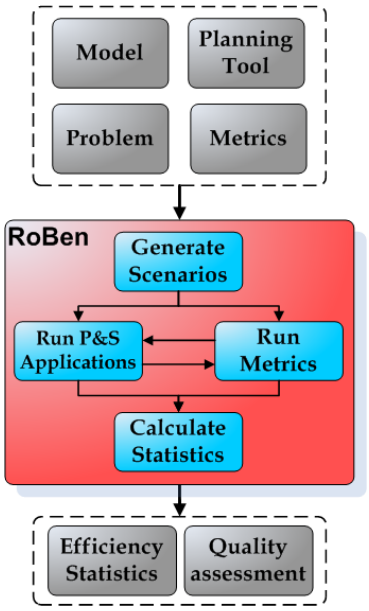

Figure 1: RoBen Architecture

2. MODELLING LANGUAGES

The problem generated via RoBen is based on the DDL3 (Domain Description Language) and the associated PDL (Problem Domain Description) [10]. This language is used in the area of AI Timeline Planning and in particular by the APSI planner [2], which is a timeline planning framework that uses Component-Based formalism.

A problem in DDL3/PDL is decomposed into components. Examples of components in the rover scenario might be simple elements (e.g., the camera of a rover), complex elements (e.g., the locomotion system composed of wheels, motors, etc), or external elements (e.g., a rock on the surface). The process of representing a planning domain/problem is then composed of the following steps:

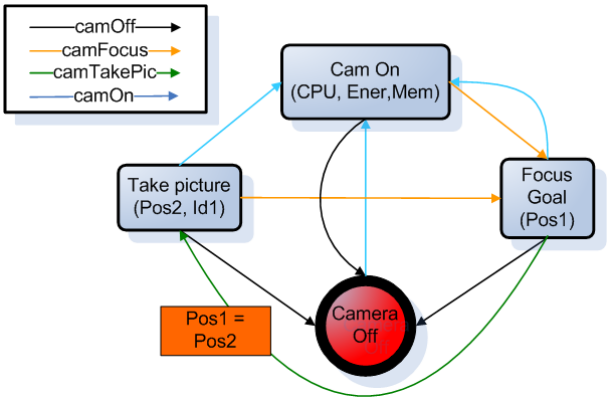

Figure 2: Automaton describing the state transitions of the camera on-board a rover

1. Modelling (components): The P&S problem is modelled by identifying the set of relevant features called *components*.

2. Synchronizing components (Domain Theory): Once all the components are created, *synchronisations* between them are created. A synchronisation describes collaborations among the components. A component may require other components to be in a specific state in order to change its state.

3. Problem description: A problem description represents a specific instance of the domain with the initial state of the world (values of the components). The goal is defined as a set of values (called *ValueChoices*) that some components must have in specific instants or periods of time.

A component is represented in DDL3 as a finite automaton (Figure 2) containing the valid state transitions. Each component has an associated *timeline* that represents the evolution of the state of the component over time, limited by a time horizon. Decisions are posted along the timeline of the components either as *ValueChoices* over the set of values of the state variable, or as *consumption/production* activities on a resource.

3. AUTONOMOUS ROVER PROBLEM

A planetary rover scenario will be used throughout the paper as an ongoing example. It is comprised of a rover equipped with a pan-tilt unit (PTU), a stereo camera (mounted on top of the PTU) and an antenna. The rover is able to autonomously navigate the environment, move the PTU, and take pictures and communicate images to a remote orbiter that is not visible for some periods.

To obtain a timeline-based specification of our robotic domain, we consider each of the above elements as a *Component*, each with its own automaton that contains a number of *ValueChoices* that represent the states of the automaton. The states can be described as follows:

- Navigation: Can be in a certain position (At(x, y)) or moving to a certain destination (GoingTo(x, y)).

- PTU: Can assume a PointingAt(pan, tilt) value if pointing in a certain direction, or a MovingTo(pan, tilt) value when it is moving.

- Camera: Can take a picture of a given object in a position <x, y> by requesting a <pan, tilt> for the PTU and a file location in the onboard memory (TakingPicture(file-id, x, y, pan, tilt)). Assumes the value CamIdle() if in an idle state.
- Antenna: Can be transmitting a given file (Communicating(file-id)) or can be idle (CommIdle()).
- Orbiter Visibility: Indicates the visibility of the orbiter. Its possible values (Visible or Not-Visible) represent external constraints for the P&S problem. In particular, these values represent contingent communication opportunities for the rover.

The rover must obey some operative rules for safety reasons. The following *constraints* must hold during the overall mission:

- While the robot is moving the PTU must be in the safe position.
- The robotic platform can take a picture only if the robot is stationary, is in one of the requested locations, and the PTU is pointing in the correct direction.
- Once a picture has been taken, the rover must send the picture to the base station.
- While communicating, the rover must be stationary.
- While communicating, the orbiter must be visible.

The system also has a set of synchronisations between ValueChoices:

- PointingAt(0, 0) value must occur during a GoingTo(x, y) value (C1).
- At(x, y) and PointingAt(pan, tilt) values must occur during a TakingPicture(pic, x, y, pan, tilt) value (C2).
- Communicating(pic) must occur after a TakingPicture(pic, x, y, pan, tilt) (C3).
- At(x, y) value must occur during a Communicating(file) (C4).
- Visible value must occur during a Communicating(file) (C5).

Figure 3 contains a representation of the system, where dotted lines represent synchronisations and normal lines represent transitions.

4. AUTOMATIC PROBLEM GENERATION

The process of generating an automated problem consists of three main steps: extracting relevant information from the model, calculating the number of occurrences of each ValueChoice using linear programming, and generating the problem in the proper format. The following inputs are required to complete these steps:

- A formal model of the system (M)
- The desired resource complexity ($T(R)^{user}$)
- The desired time complexity ($T(t)^{user}$)
- Constraints supplied by the user, specifically:
  - The maximum number of occurrences for each ValueChoice ($V_{i,j}^{max(x)}$)
  - The average execution time of each stable ValueChoice ($\bar{V}_{i,j}^{avg}$)

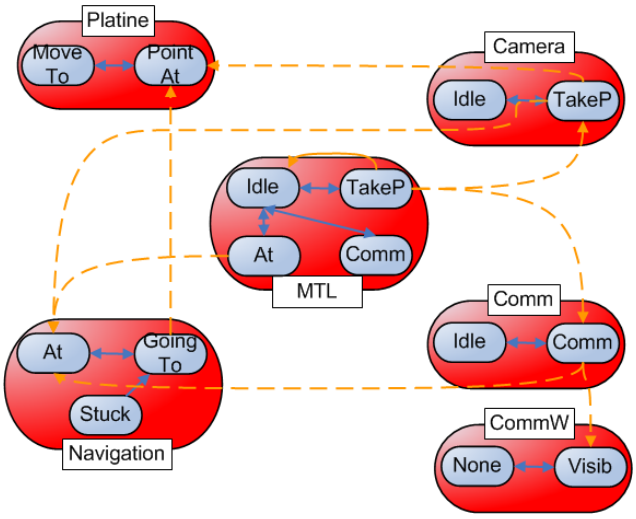

Figure 3: Rover Domain

  - The average resource consumption for each ValueChoice for each resource ($V_{i,j}^{R_k}$)

The model represents the system for which the plan will be generated. Several pieces of information are extracted from the model, including its components, component decisions, synchronisations, etc. An inclusive list of these elements are described in Section 4.1. A model of the domain described in Section 3 is used to evaluate RoBen.

The desired resource complexity is given in the range $[0, +\infty)$, where 0 represents the trivial problem, 1 represents full resource consumption, and $T(R) > 1$ represents overconsumption of the resources. The time complexity is also given in the range $[0, +\infty)$, where 0 represents the trivial problem (no goals are assigned), 1 represents full use of the available time, and $T(t) > 1$ represents the assignment of goals whose total time requirement exceeds the available time.

The user can specify constraints that limit the maximum number of occurrences for each ValueChoice in order to generate more realistic problems. Using the model presented in (3) as an example, the user might specify the following constraint:

$$Navigation.StuckAt.numOccurrences = 0 \qquad (1)$$

In this example, the user wanted to prevent the generation of problems in which the rover being stuck at location could be specified as a goal.

Additionally, the user must specify the average execution time for each stable ValueChoice. The worst-case scenario is considered where it is assumed that the execution of any ValueChoice in a component requires the execution of all other ValuesChoices in that component (i.e., cyclic transitions). As a result, the average execution time of each non-trivial ValueChoice is the same for ValueChoices in the same component. In this context, non-stable ValueChoices are called *transitional* ValueChoices. These are ValueChoices whose average execution time cannot be described and generally has no upper limit. For example, a trivial ValueChoice in the model presented in Section 3 is $Camera.Idle()$. Although a transition within the component $Camera$ may require $Idle()$ to be executed, it should not be considered when calculating the average execution time of the ValueChoices in $Camera$.

Finally, the user must specify the average resource consumption

of the ValueChoices. This is to ensure that the number of occurences of the ValueChoices assigned in the generated problem do not consume more resources than are allocated by the resource complexity. The average resource consumption is considered when specifying this constraint, which is described in greater detail in Section 4.2.

The following sections described the three main steps of the automatic problem generator.

4.1 Model Analysis

The analysis of the model consists of two steps: extraction of the model information and an analysis of the synchronisations among the ValuesChoices. The following elements are extracted from the model:

- An automaton for each component showing the transitions among its ValueChoices.
- The resources (consummable and reusable) used by the ValueChoices.
- The synchronisations of ValueChoices among different components.
- The horizon time (H).

In order to consider the fact that the execution time of a ValueChoice increases if it has synchronisations that must also be executed, the propagated time of each stable ValueChoice ($\bar{V}_{i,j}^{prop}$) must be calculated. Because the synchronisations among the ValueChoices may contain cycles, an upper limit on the number times the propagated time of a ValueChoice is updated is taken as the size of the cycle. This value is depicted as $\bar{V}_{i,j}^{up}$ and is calculated using the Tarjan graph cycle algorithm [3]. A description of the algorithm used to calculate the propagated time of every stable ValueChoice is shown in Algorithm 4.1.

Algorithm 1 The time-propagation algorithm

```
1: procedure MAIN( )
2:     update_list = V̄
3:     for all V̄_{i,j} ∈ V̄ do
4:         V̄^{prop}_{i,j} = V̄^{avg}_{i,j}
5:     while update_list ≠ ∅ do
6:         v = RemoveItem(update_list)
7:         if synchronisations(V̄_{i,j}) ≠ ∅ then
8:             UpdatePropagationTime(v)
9:             for all V̄_{i,j} ∈ V̄ − {v} do
10:                if (V̄^{up}_{i,j} > 0) & (V̄_{i,j} ∉ update_list) then
11:                    update_list.add(V̄_{i,j})

12: procedure UPDATEPROPAGATIONTIME(V̄_{i,j})
13:     for all v ∈ synchronisations(V̄_{i,j}) do
14:         V̄^{prop}_{i,j} += v^{prop}
15:     V̄^{up}_{i,j} −= 1
```

4.2 ValueChoice Occurrence Assignment

The automata describing the model of the timeline system can be transformed to a non-deterministic Turing machine. By taking this into consideration, an analysis of the time-complexity ($T(t)$) of the model can be performed, which is equal to $2^{O(t(n))}$ [15]. For the model, the length of the chain is equal to the time required to execute a set of goals, divided by the time to the horizon in order to make it proportional to the available time. This formulation is shown in (2).

$$T(t) = 2^{\frac{\sum_{j=i}^{\bar{V}_i^{num}} \bar{V}_{i,j}^{prop} \cdot \bar{V}_{i,j}^{x}}{H}} - 1 \tag{2}$$

where H is the time until the horizon is reached (defined in the model).

The number of times each value of each component must be executed in the generated problem (i.e., $V_{i,j}^{x}$ for every value in every component) was calculated using integer linear programming (ILP). The goal of this approach was to maximise the total propagated time required by the assigned $V_{i,j}^{x}$, subject to constraints derived from the desired value complexity ($T(t)^{user}$) and desired resource complexity ($T(R)^{user}$) input by the user. The special case of ILP is required over standard linear programming (LP) because all $V_{i,j}^{x}$ must belong to the set of natural numbers ($\mathbb{N}$). The formulation of the ILP is shown in (3)–(5).

Maximise:

$$z = \sum_{i=1}^{C^{num}} \sum_{j=1}^{\bar{V}_i^{num}} \bar{V}_{i,j}^{prop} \cdot \bar{V}_{i,j}^{x} \tag{3}$$

Subject to:

$$\sum_{j=1}^{\bar{V}_1^{num}} \bar{V}_{1,j}^{prop} \cdot \bar{V}_{1,j}^{x} \leq \log_2[T(t)^{user} + 1] \cdot H \tag{4}$$

$$\sum_{j=1}^{\bar{V}_2^{num}} \bar{V}_{2,j}^{prop} \cdot \bar{V}_{2,j}^{x} \leq \log_2[T(t)^{user} + 1] \cdot H$$

$$\vdots$$

$$\sum_{j=1}^{\bar{V}_{C^{num}}^{num}} \bar{V}_{C^{num},j}^{prop} \cdot \bar{V}_{C^{num},j}^{x} \leq \log_2[T(t)^{user} + 1] \cdot H$$

and

$$\sum_{i=1}^{C^{num}} \sum_{j=1}^{V_i^{num}} V_{i,j}^{R_1} \cdot V_{i,j}^{x} \leq T(R)^{user} \cdot R_1^{max} \tag{5}$$

$$\sum_{i=1}^{C^{num}} \sum_{j=1}^{V_i^{num}} V_{i,j}^{R_2} \cdot V_{i,j}^{x} \leq T(R)^{user} \cdot R_2^{max}$$

$$\vdots$$

$$\sum_{i=1}^{C^{num}} \sum_{j=1}^{V_i^{num}} V_{i,j}^{R_{R^{num}}} \cdot V_{i,j}^{x} \leq T(R)^{user} \cdot R_{R^{num}}^{max}$$

Where:

$$V_{i,j}^{x} \in \{0, 1, 2, \ldots, V_{i,j}^{max(x)}\}$$

The value complexity constraints in (4) are a reconfiguration of (2) for each component. These constraints prevent the value complexity of each component ($T(t)_i^{prob}$) from exceeding the desired value complexity input by the user ($T(t)^{user}$). The overall value complexity of the generated problem is simply taken as

$$T(t)^{prob} = \frac{\sum_{i=1}^{C^{num}} T(t)_i^{prob}}{C^{num}}, \tag{6}$$

which is the average complexity over all the components. The resource complexity constraints in (5) prevent the assignment of $\bar{V}_{i,j}^{x}$ that would cause overconsumption for any resource, which is the

product of the desired resource complexity input by the user and the maximum capacity of each resource, as defined in the model.

The ILP was solved using the open source GNU Linear Programming Kit (GLPK)[1]. This solver uses an optimized version of the branch and bound method. The output of the solver is passed to the problem generator described in Section 4.3.

4.3 Problem Generation

The output of the ILP system is used to generate the problem in PDL. Each of the occurrences of each ValueChoice, adds a goal to the file and the parameter values are filled in invoking the special procedures. The following paragraph shows a goal generated by RoBen consisting of taking a picture.

```
g20 <goal> Camera.camera.TakingPicture(?file_id1 = 1, ?x1 = 2, ?y1 = 1, ?pan1 = 10, ?tilt1 = 40) AT [0, +INF] [1, +INF] [1, 100];
```

The initial conditions of the system are also represented in the PDL file via facts that, in opposition to goals, do not need to be justified by the planner. Therefore, RoBen establishes as initial conditions default values chosen from the domain. The PDL file, together with the DDL represent the inputs to be passed to the planner.

5. PROBLEM EVALUATION

In order to validate the heuristic, $T(t)^{prob}$ can be compared with the performance of a planner to understand whether an increasing value of $T(t)^{prob}$ also represents a more complex search space with less solutions. Increasing this value represents a higher number of goals that should reduce the potential number of solutions. We are also interested in understanding the relation between $T(t)^{prob}$ and the final percentage of time demanded for each timeline. We have performed several tests using the APSI planner [2], consisting on the generation of problems with increasing $T(t)^{prob}$ in the range $[0.1 - 1]$, limiting the execution time to 30 minutes for the rover domain presented in Section 3.

The constraints defined for the system are:

- One constraint of type (4) for each component of the domain
- Semantic constraints to define the list of transitional states: $\tilde{V}$ = {Navigation: (GoingTo, StuckAt), Platine: (MovingTo), Camera: (CamIdle), Antenna: (CommIdle), CommunicationVW: (All), MissionTimeline: (All)}

The results generated by RoBen are shown in Table 1.

It is important to remark that the error shown is inherent to the linear programing. The fact that the ILP tries to allocate an integer number of tasks that consumes a total time smaller or equal than the maximum time available (the Horizon) limits the number of solutions. This constraint becomes critical in case the time required by a ValueChoice is close to the Horizon time or $T(t)^{user}$ is too small. An example of the last situation can be observed in the table for $T(t)^{user} = 0.1$. The ILP is not able to generate a problem close to this complexity because the allocation of one single $V_{i,j}$ for almost all components makes $T(t)^{prob} > T(t)^{user}$.

Figure 4 represents the number of goals and variables relative to each of the problems generated. The results of the executions displayed in Figure 5 show an increasing complexity in terms of time required by APPlanner to solve the problem directly related to the increase of $T(t)^{prob}$. Notice that the planner was not able to find a solution for $T(t)^{prob} = 0.9$ and $T(t)^{prob} = 1$ in less that 30 minutes. Early analysis of these results lead us to think that the heuristic based on $T(t)$ produce problems with appropriate complexities.

6. CONCLUSIONS AND FUTURE WORK

In this paper we described a benchmarking tool called RoBen. The main objective of RoBen is to provide a means to evaluate and compare automated planners. RoBen, given a specific domain as

[1]http://www.gnu.org/software/glpk/

$T(t)^{user}$	Navi-At	Platine-PointAt	Camera-TakePic	Antenna-Comm	Error (%)
0	0	0	0	0	0
0.1	0	2	0	0	82
0.2	1	5	0	0	57.75
0.3	1	7	1	1	28.5
0.4	2	9	1	0	30
0.5	2	11	2	2	12.7
0.6	3	13	2	2	14.7
0.7	3	15	2	2	22.85
0.8	4	16	2	2	23.63
0.9	4	18	3	3	10.75
1	5	20	3	3	9.85

Table 1: ValueChoices ocurrence assignment

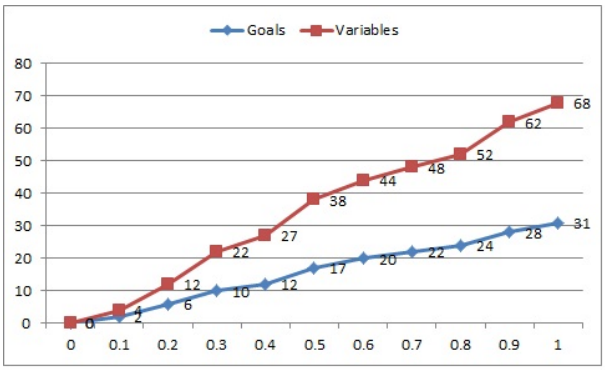

Figure 4: Number of goals and variables respect complexity

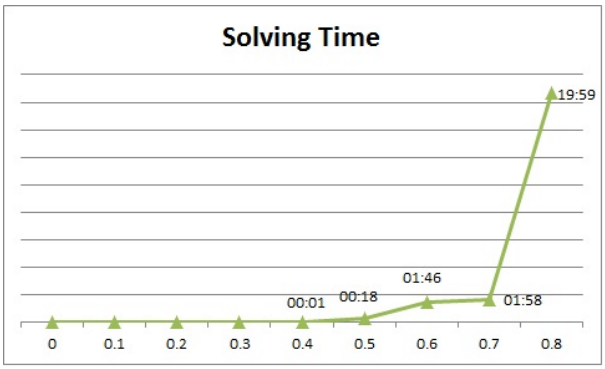

Figure 5: Planner solving time respect complexity

input, generates sets of problems which can be used to evaluate the quality of P&S algorithms.

Regarding the results obtained, it is important to remark that it turned out to be a difficult enterprise to generate actual valid plans due to hidden incongruencies in constraints that were preventing the planner in finding a solution. After some tuning of the PDLs, it was possible to find solutions, but it requires some knowledge of the domain that has not yet been automated. It was also difficult to evaluate the results provided by the planner apart from the execution time. Due to the fact that the planner is generating flexible timelines and that the execution of some $V_{i,j}$ cannot be estimated, like it happens for most of the $\tilde{V}$ ValueChoices, it is difficult to compare the plan with $T(t)^{prob}$. Presently, the analysis is focused on the planning time, but random assigments of execution times to all the $V_{i,j}$ in the domain might provide new indicators. However, they must be added carefully in order to avoid incongruencies or 0-solution search spaces.

Future work will consider different alternative evolutions of RoBen. A first direction is to introduce further metrics in order to evaluate the completeness of a benchmark set. In other words, given a domain model, the objective is to verify that all the aspects in the domain are covered and stressed (i.e., according to a set of requirements). Regarding the domain language, the results obtained so far can lead to further evolution of the modeling languages. In fact, as a side effect of the empirical evaluation, we noticed some limitations (and possible extensions) of DDL3 and PDL. As an example, it would be convenient to add semantic information, at least in the domain language, to provide meta-information about states and constraints such as average execution time, type of state (error, stable or transitional), etc.

Acknowledgments. This research has been co-funded by the Networking/Partnering Initiative (NPI) in collaboration between ESA-ESOC and TU Darmstadt. It also receives support from the German Research Foundation (DFG) within the Research Training Group 1362 “Cooperative, adaptive and responsive monitoring in mixed mode environments". The authors would like to thank Simone Fratini for his valuable support

7. REFERENCES

[1] J. Bresina, A. Jónsson, P. Morris, and K. Rajan. Mixed-initiative activity planning for Mars rovers. In *Proceedings of the 19th international joint conference on Artificial intelligence*, IJCAI'05, pages 1709–1710, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[2] A. Cesta and S. Fratini. The timeline representation framework as a planning and scheduling software development environment. In *In PlanSIG-08, Proceedings of the 27th Workshop of the UK Planning and Scheduling Special Interest Group*, 2008.

[3] B. V. Cherkassky, K. St, and A. V. Goldberg. Negative-cycle detection algorithms. *Mathematical Programming*, 85:349–363, 1996.

[4] S. Chien, R. Doyle, A. Davies, A. Jonsson, and R. Lorenz. The Future of AI in Space. *Intelligent Systems, IEEE*, 21(4):64 –69, july-aug. 2006.

[5] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using Iterative Repair to Improve Responsiveness of Planning and Scheduling. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, pages 300–307, 2000.

[6] S. Chien, R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castaño, A. Davies, D. Mandl, S. Frye, B. Trout, J. D'Agostino, S. Shulman, D. Boyer, S. Hayden, A. Sweet, and S. Christa. Lessons learned from autonomous sciencecraft experiment. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, pages 11–18, New York, NY, USA, 2005. ACM.

[7] A. G. Davies, S. Chien, T. Doggett, F. Ip, and R. C. no. Improving Mission Survivability and Science Return with Onboard Autonomy. In *In Proc. of 4th International Planetary Probe Workshop (IPPW)*, 2006.

[8] ECSS. *ECSS-Q-80-04 – Guidelines for Software Metrication Programme Definition and Implementation*. European Cooperation for Space Standardization (ECSS), February 2006.

[9] ECSS. *ECSS-Q-ST-80C – Space product assurance - Software product assurance*. European Cooperation for Space Standardization (ECSS), March 2009.

[10] S. Fratini, F. Pecora, and A. Cesta. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. *Archives of Control Sciences*, 18(2):231–271, 2008.

[11] D. Long and M. Fox. The International Planning Competition Series and Empirical Evaluation of AI Planning Systems, 2006.

[12] N. Muscettola. HSTS: Integrating Planning and Scheduling. In M. Zweben and M. S. Fox, editors, *Intelligent Scheduling*, pages 169–212. Morgan Kaufmann, 1994.

[13] N. Muscettola, P. P. Nayak, B. Pell, and B. C. Williams. Remote Agent: to boldly go where no AI system has gone before. *Artificial Intelligence*, 103:5–47, August 1998.

[14] A. Orlandini, A. Finzi, A. Cesta, S. Fratini, and E. Tronci. Enriching APSI with Validation Capabilities: The KEEN environment and its use in Robotics. In *Proceedings of the 11th ESA Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, 2011.

[15] M. Sipser. *Introduction to the theory of computation*. Computer Science Series. Thomson Course Technology, 2006.

APPENDIX – Nomenclature

C_i	Component i
C^{num}	Number of components
H	Horizon time
M	Model
P	Planning tool
V	Set of all ValueChoices for all components
$V_{i,j}$	ValueChoice j of C_i
$V_{i,j}^{max(x)}$	Max number of times $V_{i,j}$ can be executed
V_i^{num}	Number of ValueChoices in C_i
$V_{i,j}^{R_k}$	Average quantity of R_k consumed by $V_{i,j}$
$V_{i,j}^{sync}$	Number of ValueChoices syncronised with $V_{i,j}$
$V_{i,j}^{x}$	Number of times $V_{i,j}$ must be executed
$\bar{V}$	Set of all stable ValueChoices for all components
$\bar{V}_{i,j}$	ValueChoice j of C_i (stable)
$\bar{V}_{i,j}^{avg}$	Average time required to execute $V_{i,j}$
$\bar{V}_i^{num}$	Number of stable ValueChoices in C_i
$\bar{V}_{i,j}^{prop}$	Propagated time required to execute $V_{i,j}$
$\bar{V}_{i,j}^{up}$	Number of times $\bar{V}_{i,j}^{prop}$ can be updated
$\tilde{V}$	Set of all transitional ValueChoices for all components
$\tilde{V}_{i,j}$	ValueChoice j of C_i (transitional)
R_k	Resource k
R_k^{max}	Maximum capacity of R_k
R^{num}	Number of resources
$T(R)^{prob}$	Resource complexity of the generated problem
$T(R)^{user}$	Resource complexity input by the user
$T(t)^{prob}$	Time complexity of the generated problem
$T(t)_i^{prob}$	Time complexity of C_i in the generated problem
$T(t)^{user}$	Time complexity input by the user