RoBen: Introducing a Benchmarking Tool for Planetary Rover Planning & Scheduling Algorithms

J. M. Delfa[*]

Department of Computer Science Technische Universität Darmstadt, Darmstadt, 64289, Germany

N. Policella and M. Gallant and A. Donati and R. Bertrand

ESA-ESOC, Darmstadt, 64293, Germany

O. von Stryk

Department of Computer Science, Technische Universität Darmstadt, Darmstadt, 64289, Germany

Y. Gao

Surrey Space Center, University of Surrey, Guildford, GU2 7XH, United Kingdom

Automated Planning & Scheduling Systems are nowadays applied in a wide range of spacecraft, from satellites to Mars rovers. The planner is responsible for the generation of valid plans that determine the activities to be performed by the spacecraft, given a set of goals and constraints (the problem), and taking into consideration the status of the spacecraft and environment. Therefore, it represents a critical system that needs to be strictly validated and verified.

This paper presents a benchmarking tool called RoBen intended to characterize the performance of timeline planning systems. Using a number of metrics and heuristics, RoBen can generate synthetic problems of a given complexity in order to stress planners at different levels. At the same time, we are looking for properties that could help us to determine when a problem is unsolvable.

[*]Also affiliated to Surrey Space Centre, University of Surrey (UK) and ESA-ESOC, Darmstadt, 64293, Germany

Nomenclature

C_i	Component i
C^{num}	Number of components
H	Time horizon
M	Model
P	Planning tool
V	Set of all ValueChoices for all components
$V_{i,j}$	ValueChoice j of C_i
V_i^{num}	Number of ValueChoices in C_i
$V_{i,j}^{x}$	Number of times $V_{i,j}$ must be executed
$V_{i,j}$	ValueChoice j of C_i
$V_{i,j}^{inst}$	Instant time required to execute $V_{i,j}$
$V_{i,j}^{prop}$	Propagated time required to execute $V_{i,j}$
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

I. Introduction

Recently, robotics has been gaining prominence in several space scenarios such as planetary exploration or ISS exploitation. 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,[1,6,13,14] 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.[10] 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.[15]

This work focused on understanding the intrinsic properties of timeline-type problems. This in our opinion should allow the complexity of the associated planning/solving process to be determined regardless of the specific planning system. Understanding these properties would have two important consequences:

- An estimation of whether a problem has a solution before actually solving it, which is important as some problems require powerful computational systems running for long periods of time.
- A characterization of the performance of timeline-planners under different levels of stress. This would help to understand the weak points of the planner and to identify the best one to solve a given problem.

Problem complexity metrics can be divided between:

- Static metrics that only consider the problem input and initial 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.

RoBen represents a mission-independent benchmarking tool aimed to provide a standard framework to evaluate and compare timeline-based planning tools as well as helping future operators validate alternative problems and plans. In particular, RoBen automatically generates problems to evaluate the performance of P&S algorithms for rover scenarios. It provides a set of static metrics which are 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 dependent on the specific characteristics of the mission and/or planning tools. Therefore, a combination of metrics both novel and borrowed from the literature[5,9,12] have been used to improve the results.

This work represents an evolution of the system presented on.[8] We have focused the efforts in the refinement of problem complexity estimation, which led us to reconsider some of the assumptions and statements cited in the previous paper. A GUI has been implemented to streamline the execution of experiments.

II. Planning formalism

This work is oriented toward timeline planning systems, which involve a number of concepts that will be used throughout the paper. We use APSI[2] as our reference timeline planning system. In APSI, the process of representing a planning domain/problem is decomposed of the following steps:

1. Modeling (components): The system is modeled as a set of *components* that represent features of the system. 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 model is described with a formal language called DDL3 (Domain Description Language).[11] There are two type of components in APSI:

    - StateVariable: Represents a subsystem with some functionality, such as a camera for example. It is formalized in DDL3 as a finite automaton (Figure 1) containing a set of states called ValueChoices and a set of transitions between them.
    - Resource: Represents an element that can be consumed or produced, such as energy. They are formalized by the maximum capacity and the current status.

    ValueChoices and Consumption/Production actions represent **decisions**. Each component has associated a *timeline* that represents the evolution of the state of the component over time, limited by the time horizon. Decisions are posted along the timeline describing the changes on the status of the component along the plan time

2. Synchronizing components (Domain Theory): Once all the components are created, *synchronizations* between them are created. A synchronization is used to express dependencies between components and are defined together with the components in the model of the system.

3. Problem description: A problem represents a specific instance of the domain and is defined in terms of goals and initial conditions. Goals are defined as ValueChoices that some components must achieve at specific instants or periods of time, while initial conditions represent the status of the system at the starting time. Problems are formally described using PDL (Problem Description Language).

III. 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.

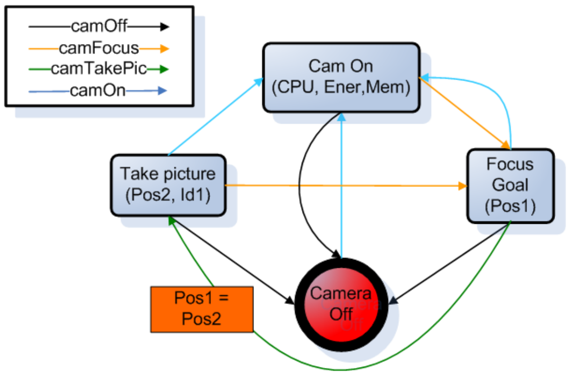

Figure 1. Automaton describing the state transitions of the camera on a rover

- 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 on-board 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 synchronizations 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 2 contains a representation of the system, where dotted lines represent synchronizations and normal lines represent transitions.

IV. Generation of synthetic Problems

The workflow of RoBen is depicted on Figure 3. The user must first generate a number of inputs including the model (Section II) and the desired problem complexity. RoBen then generates a synthetic problem using a process that is subdivided into 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 PDL format. The problem

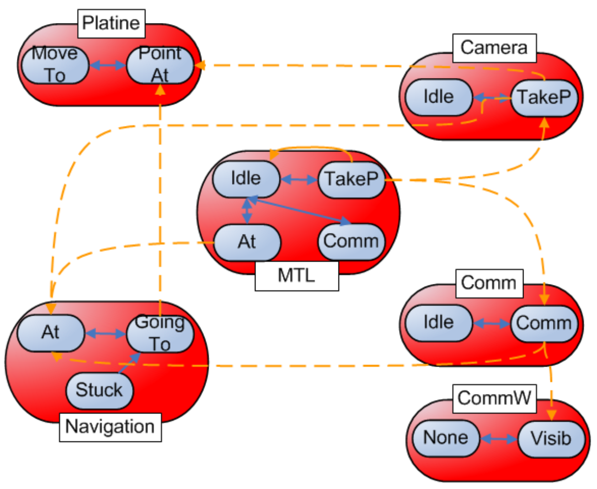

Figure 2. Rover Domain

is then injected into the planning algorithm to be analyzed together with the model and some metrics are extracted during the solving process.

The key element stems on the accuracy to estimate the problem complexity. We have to focus on intrinsic properties of the problem itself independent of the planning system. We have identified the following:

- Timeline fragmentation: Directly related to the number of goals to be allocated per timeline, it is a good indicator of the effort a timeline-based planner needs to create a valid plan.
- Timeline occupancy: Relation between the estimated time to execute the goals of the problem with respect the time horizon available in the plan for these goals. It is a good indicator of the number of solutions in the search space.

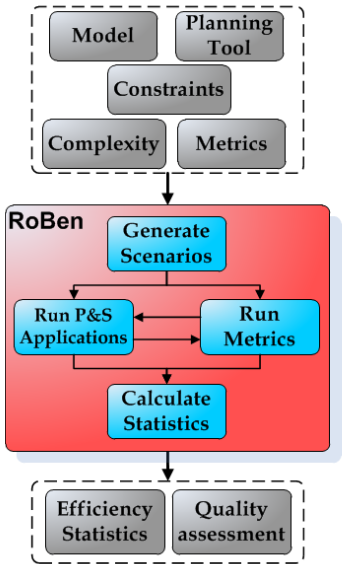

Figure 3. RoBen Architecture

In the following sections, we outline the assumptions required to use RoBen and the inputs that are necessary for its execution. A detailed description of the system follows.

A. Assumptions

As with any heuristically guided algorithm, RoBen makes some assumptions in order to estimate the problem complexity, which are enumerated bellow:

- [General rule]: Problems with a higher demand of time are considered more difficult.
- [General rule]:Problems with a greater fragmentation of the timeline (i.e., greater number of goals and required supporting activities) are considered more difficult.

1 After achieving a goal g (ValueChoice), the automaton returns to its initial state is (a different ValueChoice). Transitions among the ValueChoices always follow the shortest path.

2 None of the constraints related to the goal are satisfied.

3 Given a time interval $[l, u]$ for the duration of each state and transition, the time required for a component to transition from its initial to goal ValueChoice is the lower bound l for all time intervals in the path (the minimal duration).

B. Requirements

The user must provide a number of inputs to the system:

- A formal model of the system (M): In this paper, we have used the model presented in Figure 2
- The desired resource complexity ($T(R)^{user}$): 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 desired time complexity ($T(t)^{user}$): 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.
- User defined constraints:
  - Maximum number of occurrences for each ValueChoice ($V_{i,j}^{max(x)}$).
  - An interval $[l, u]$ with the lower and upper execution times for the ValueChoice ($V_{i,j}$).
  - A label indicating the default ValueChoice of a component. There must be only one default ValueChoice in order to estimate the propagated time to reach each of the states of the automaton.
  - Labels indicating whether ValueChoices are nominal. Non-nominal states like $RobotBase - Stuck$ are not taken into account during the estimation of the propagated time.
  - The average resource consumption for each ValueChoice for each resource ($V_{i,j}^{R_k}$): Used to ensure that the number of occurrences of the ValueChoices assigned in the generated problem do not overconsume the resources. The average resource consumption is considered when specifying this constraint, which is described in greater detail in Section E.
- Initial conditions of the components (facts).

The user can limit the maximum number of occurrences for each ValueChoice in order to generate more realistic problems. For example, the following constraint:

$$Navigation.StuckAt.numOccurrences = 0$$

will ensure the non-nominal ValueChoice StuckAt (the rover cannot move) will not be assigned in the problem. This ValueChoice is used for contingency purposes during execution time and won't appear in the plan.

C. Problem Complexity

The overall value complexity of a problem is calculated according to the following equation:

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

where:

C^{num}	Number of components
$T(t)_i^{prob}$	Time complexity of component C_i in the generated problem

This work focus on the estimation of StateVariables complexity. Even though it is foreseen to extend the model in order to estimate resource complexity, further work should be done in this direction.
The automata describing the StateVariables of a model can be represented as non-deterministic Turing machines (NTM). By taking this into consideration, an analysis of the time-complexity ($T(t)$) of the model can be performed, which is equal to $2^{O(t(n))}$.[16] For our model, the length of the input chain in a NTM is equal to the time required to execute all the goals with respect to the time horizon. This formulation is shown in (2).

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

where:

V_i^{num}	Number of nominal ValueChoices in C_i
$V_{i,j}^{prop}$	Propagated time required to execute $V_{i,j}$
$V_{i,j}^{x}$	Number of times $V_{i,j}$ must be executed
H	Time horizon

D. Calculation of propagated time $V_{i,j}^{prop}$

A difficulty with equation (2) is that the time that a given goal g will require in the plan is unknown. It is the mission of the planner to provide a solution that will bound this time in a range with a lower and upper limit $[l, u]$. Therefore, a heuristic is required to estimate this time that we will call propagated time ($V_{i,j}$).

The heuristic to estimate the propagated time for a given goal g should take into consideration the effect of the internal transitions on its own component to achieve g plus the propagated time required by supporting actions required by g from other components. For example, in Figure 2, to estimate the time required to take a picture, it is required to estimate the time required for the component Camera to take the snapshot plus the time required by the Navigation system to be positioned in the right place. A valid heuristic should generate propagated times for each ValueChoice within the range between the lower and upper bound given by the planner to this ValueChoice in the timeline. Therefore, we based our heuristic on the assumptions [1] to [3] cited in Section A, which represents an above worst-case scenario. These assumptions represent an important change with respect to the previous version in which the longest-path with the maximum duration was used. As a result, the propagated time estimation has decreased and therefore the number of goals allocated in the synthetic problem has increased.

Bellow, each step to estimate $V_{i,j}$ is described in detail.

1. Instant time estimation $V_{i,j}^{inst}$

First, the time required for a component to get from its initial state is to a different state d is estimated as follows:

$$V_d^{inst} = shortest\_path(V_{is}, V_d) + shortest\_path(V_d, V_{is}) \tag{3}$$

For each decision d of each component C_i, an estimation is made of the minimal shortest path from is to d and back to is, which represents the shortest path (avoiding loops) taking into consideration the minimal possible duration for each temporal constraint found in the path. It represents a more realistic and accurate method than the one used previously, in which an average time was calculated for all decisions of the same component.

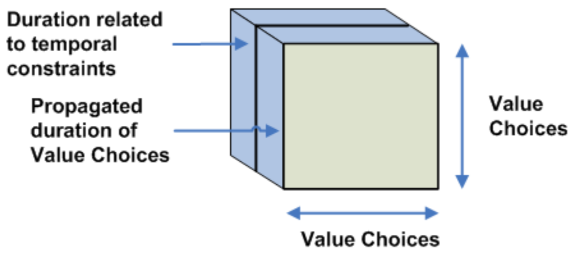
Duration related to temporal constraints
Propagated duration of Value Choices
Value Choices
Value Choices

Figure 4. Propagation Matrix

2. *Population of the Propagation Matrix PM*

Next, a square matrix called the Propagation Matrix or PM is generated (see Figure 4) in order to calculate the propagated time of each decision.

Each decision for each component of the model is represented as a row and column with the same index. Each cell of the diagonal in the matrix, $PM[i,i]$, contains the propagated time for the corresponding decision. The rest of the cells, $PM[i,j]|i \neq j$ contain the propagated time of decision j in case i needs it as a supporting activity, or 0 if it is not needed. The matrix contains a third dimension to store the propagated duration of the temporal constraints associated with each transition.

Initially, the diagonal is populated with the instant time for each decision $V_{i,j}^{inst}$:

$$PM[j][j][0] = V_i^{inst} | \forall C_i, \forall V_{i,j} PM[j][j][1] = 0 \tag{4}$$

The first assignment, $PM[j][j][1] = 0$, represents the fact that the transitions between states[a] of the same component are assumed to be instantaneous.

Once the diagonal is populated, the synchronizations between decisions of different components are taken into consideration. Given a decision i that is synchronized with another decision j, the matrix will be updated as follows:

$$PM[i][j][0] = V_j^{inst} PM[i][j][1] = \text{Temporal relation duration } (trd) \tag{5}$$

For each decision j with which i synchronizes, the average duration of j is added to $PM[i][j][0]$. The value trd is calculated in a different way depending on the temporal relation associated to the synchronization (just Interval-Interval are of interest here). The following list displays the equations for each of the relations:

- $i\ EQUALS\ j$: $PM[i][j][1] = 0$
- $i\ BEFORE/AFTER\ j$:

$$PM[i][j][1] = \begin{cases} (l+u)/2 & \text{if } u \neq +\infty \\ l & \text{otherwise} \end{cases} \tag{6}$$

- $i\ MEETS/METBY\ j$: $PM[i][j][1] = 0$
- $i\ OVERLAPS/OVERLAPPEDBY\ j$:

$$PM[i][j][1] = \begin{cases} -(l+u)/2 & \text{if } u \neq +\infty \\ l & \text{otherwise} \end{cases} \tag{7}$$

- $i\ DURING/CONTAINS\ /STARTS/FINISHES\ j$: $PM[i][j][1] = 0$

As a general rule, if the temporal relation does not impose a delay between decisions i and j, a value 0 is assigned to $PM[i][j][1]$. Otherwise, as it happens with $BEFORE[l,u]$, it is assigned the mean of the lower and upper bound (l and u respectively). Notice that in case u is infinite (there is no upper bound duration), the lowest value will be used. In the case of $OVERLAPS/OVERLAPPEDBY$, this value is negative, as it has the opposite effect than $BEFORE/AFTER$.

[a]do not confuse with the duration of each state

3. *Time propagation*

Once the matrix is populated, the values will be propagated in two steps: first for the values of the synchronizations ($i \neq j$) and then for the propagated duration of each decision ($i = j$).

The value related to the duration of each supporting decision is propagated according to the temporal relation as follows:

- $i\ EQUALS\ j$: No change
- $i\ BEFORE/AFTER\ j$: $PM[i][j][0] = PM[i][j][0] + PM[i][j][1]$
- $i\ MEETS/METBY\ j$: No change
- $i\ OVERLAPS/OVERLAPPEDBY\ j$: $PM[i][j][0] = PM[i][j][0] + PM[i][j][1]$
- $i\ DURING/CONTAINS\ /STARTS/FINISHES\ j$: $PM[i][j][0] = Max(PM[i][i][0], PM[j][j][0])$

In some relations, no propagation is required as there is no time delay added by the temporal relation. In the case of $BEFORE/AFTER$ and $OVERLAPS/OVERLAPPEDBY$, the estimated temporal relation duration is added to the propagated duration of the synchronization (remember that $OVERLAPS/OVERLAPPEDBY$ have negative values). Finally, for $DURING/CONTAINS$ and $STARTS/FINISHES$ the synchronization duration will be the biggest propagated durations between i and j.

Once the synchronizations are updated, the estimated duration for each decision is propagated as follows:

$$PM[i][i][0] = \sum_{j=1}^{V^{num}} PM[i][j][0] \tag{8}$$

It means that the estimated duration of a decision will be equal to its duration plus the estimated duration of the decisions with which its synchronize.

4. *Iteration*

The graph that represents the model might contain cycles. As a consequence, the propagation algorithm could have infinite loops. As an example, imagine that the Camera requires the Locomotion system to take a picture and the Locomotion system requires a picture to navigate toward the target. In our PM matrix it will be represented as values bigger than zero in the cells $PM[i][j]$ and $PM[j][i]$. Every time the value of i is updated, it is propagated to reflect its effect in j and vice versa. To avoid this problem, the maximum number of times a value can be updated is equal to the size of the cycle. This value is named $V_{i,j}^{up}$ and is calculated using the Tarjan graph cycle algorithm.[3]

E. ValueChoice Occurrence Assignment

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) is calculated using integer linear programming (ILP). The goal of this approach is to maximize 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 (9)–(11).

$$\text{Maximize:}$$
$$z = \sum_{i=1}^{C^{num}} \sum_{j=1}^{V_i^{num}} V_{i,j}^{prop} \cdot V_{i,j}^{x} \tag{9}$$
$$\text{Subject to:}$$
$$\sum_{j=1}^{V_1^{num}} V_{1,j}^{prop} \cdot V_{1,j}^{x} \leq \log_2[T(t)^{user} + 1] \cdot H \tag{10}$$
$$\sum_{j=1}^{V_2^{num}} V_{2,j}^{prop} \cdot V_{2,j}^{x} \leq \log_2[T(t)^{user} + 1] \cdot H$$
$$\vdots$$
$$\sum_{j=1}^{V_{C^{num}}^{num}} V_{C^{num},j}^{prop} \cdot V_{C^{num},j}^{x} \leq \log_2[T(t)^{user} + 1] \cdot H$$
$$\text{and}$$
$$\sum_{i=1}^{C^{num}} \sum_{j=1}^{V_i^{num}} V_{i,j}^{R_1} \cdot V_{i,j}^{x} \leq T(R)^{user} \cdot R_1^{max} \tag{11}$$
$$\sum_{i=1}^{C^{num}} \sum_{j=1}^{V_i^{num}} V_{i,j}^{R_2} \cdot V_{i,j}^{x} \leq T(R)^{user} \cdot R_2^{max}$$
$$\vdots$$
$$\sum_{i=1}^{C^{num}} \sum_{j=1}^{V_i^{num}} V_{i,j}^{R_{R^{num}}} \cdot V_{i,j}^{x} \leq T(R)^{user} \cdot R_{R^{num}}^{max}$$
$$\text{Where:}$$
$$V_{i,j}^{x} \in \{0, 1, 2, \ldots, V_{i,j}^{max(x)}\}$$

The value complexity constraints in (10) 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

$$T(t)^{prob} = \frac{\sum_{i=1}^{C^{num}} T(t)_i^{prob}}{C^{num}}, \tag{12}$$

which is the average complexity over all the components. The resource complexity constraints in (11) prevent the assignment of $V_{i,j}^{x}$ 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. Note however that in our example, the constraints in (11) are not used (no resources, only temporal constraints).

The ILP is solved using the open source GNU Linear Programming Kit (GLPK)[b]. 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 F.

F. 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.

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

```
    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.

V. Results

In order to validate the heuristic, $T(t)^{prob}$ has been 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. The old and new version have been also compared in order to understand the effect of the modifications. 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 III.

The constraints defined for the system are:

- One constraint based on (10) 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 in Table 1 have been obtained with the original version of RoBen.

$T(t)^{user}$	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 occurrence assignment

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 time horizon 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 5 represents the number of goals and variables relative to each of the problems generated. The results of the executions displayed in Figure 6 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.

We have reproduced the same test with the new heuristic. The propagated time is now much smaller due to the assumption of minimal shortest path. The effect of the inclusion of temporal relations does not play a major role in the case of the rover model, as only one synchronization (Taking Picture $\rightarrow$ Communicating) adds delays. On the other hand, each activity requires less time in the timeline, a fact that allows for more goals in the generated plans, but with greater fragmentation. As a consequence, the number of goals generated for the synthetic problem and the time

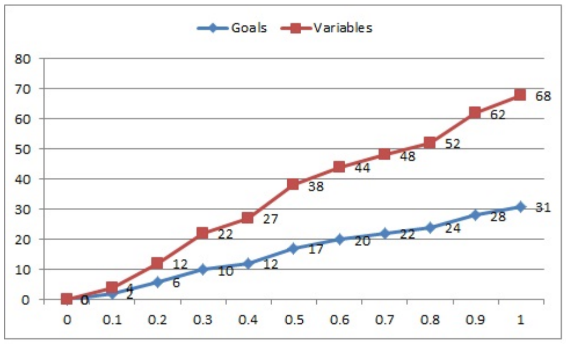

Figure 5. Number of goals and variables respect complexity

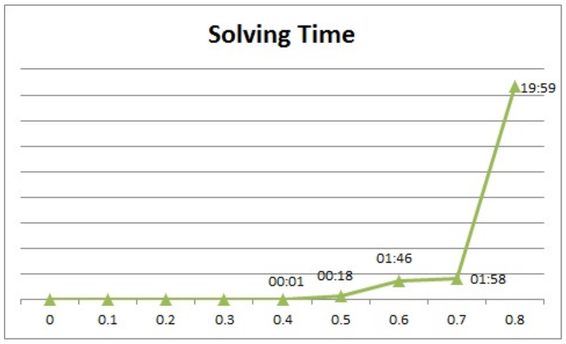

Figure 6. Planner solving time respect complexity

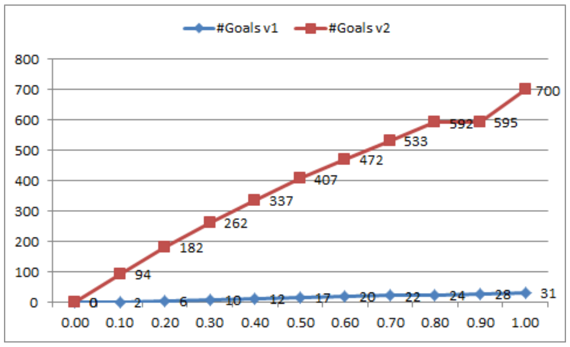

Figure 7. Comparison of Propagation Time heuristics

required by the planner have dramatically increased. Figure 7 shows a comparison between the number of goals using the two different heuristics.

VI. Conclusion

RoBen represents an effort to study the performance of timeline planning algorithms. Moreover, it tries to understand the properties that make a problem difficult to be solved. As a result, the timeline fragmentation and occupancy have been identified as relevant properties which have been analyzed using a heuristic that tries to estimate the problem complexity.

After solving some problems experienced in a previous version that prevented in some cases the generation of valid plans, the heuristic was evolved in three different directions: calculation of the instant time for each decision respect an initial state, estimation based on the minimal shortest path, and analysis of the temporal relations involved in the synchronizations. The synthetic problems generated with the new heuristic are much more complex, the planner was not able to find a solution in many cases without a full expansion of the search tree.

Future work will be dedicated to study the new heuristic under different models and planners (if possible), and characterizing the error with respect to the estimation of the propagated time. We aim also to predict the problem complexity and whether it has solutions. With respect to the modeling languages, 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 initial state, average execution time, type of state (error, nominal), 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

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 27^{th} 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] J. Delfa, N. Policella, M. Gallant, O. von Stryk, A. Donati, and G. Y. Metrics for planetary rover planning & scheduling algorithms. In *In Proc. of the Workshop on Performance Metrics for Intelligent Systems (PerMIS12)*, 2012.

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

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

[11] 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.

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

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

[14] 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.

[15] 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.

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