IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

Satisfying Resource Constraints in Space Missions by On-line Task Reconfiguration

This is the author's manuscript

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/129613 since

Publisher:

ESA (European Space Agency)

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

18 July 2024

SATISFYING RESOURCE CONSTRAINTS IN SPACE MISSIONS BY ON-LINE TASK RECONFIGURATION

Roberto Micalizio, Enrico Scala, and Pietro Torasso

Dipartimento di Informatica, Università di Torino, corso Svizzera 185, 10149 Torino, Italy

ABSTRACT

This paper addresses the problem of robust plan execution in the context of exploration mission plans carried on by planetary rovers. In order to deal with limited computational resources, and with limited rover's autonomy (the rover is not allowed to change its plan), the paper proposes a novel methodology to cope with contingencies rising at execution time. The approach aims at reconfiguring on the fly the plan by changing the parameter configuration of the actions still to be executed. By exploiting an enriched action representation based on the notion of *execution modalities*, the ReCon system is able to (i) early detect the violation of mission resource constraints, (ii) find (if any) a new assignment of modalities to preserve the mission plan constraints. ReCon uses constraint satisfaction techniques both for plan consistency and plan reconfiguration. Finally, the paper reports a preliminary set of experimental results which in practice confirms the feasibility and the advantages of the approach.

Key words: Robust plan execution; planetary rovers; CSP; plan reconfiguration.

1. INTRODUCTION

Space exploration missions carried on by planetary rovers are very challenging. On the one hand, the environment is just partially observable and very loosely predictable; thereby the rover must present some form of autonomous behavior in order to react to unexpected contingencies. On the other hand, the rover's autonomy is typically bounded both because of limitations of on-board computational power, and because the rover is not in general allowed to replace the plan synthesized on Earth.

In recent years, robust plan execution in partially known environments has been addressed in a number of works [GS10, vdKdW05, GGO10]. Most of these approaches include a replanning step: whenever unexpected situations cause the failure of an action, the plan execution is stopped and a new plan is synthesized as a result of a new planning phase. In the challenging scenario of space exploration, however, re-planning is not always possible. In [BWW06, CSW09] the replanning is avoided by means of contingent plans that encode choices between functionally equivalent sub-plans. At execution time, the plan executor is able to select a contingent plan according to the current contextual conditions. These works, however, are mainly focused on the temporal dimension and do not consider consumable resources.

In a previous work ([MST12]), we have presented a supervisor, called ActS, which monitors the execution of each mission task: whenever an anomalous execution trend is detected, ActS tries to avoid the task failure by adjusting its parameters (e.g., during a locomotion task, ActS can intervene by increasing/decreasing the rover's speed). ActS is effective in monitoring the execution of a single task, but its local decisions may have an impact on the whole mission (e.g., slowing down a locomotion may cause a delayed execution of all the following tasks). Therefore, it may be possible that the ActS's local decisions could threat some global constraints on time and resources. To mitigate the impact of such threats we propose to complement ActS with another module, called ReCon: whenever a significant discrepancy between expected and actual resource consumption is detected, ReCon reconfigures the mission tasks still to be performed in order to assure the satisfaction of the resource constraints.

More precisely, in this paper we consider the mission plan as an ordered sequence of actions; in addition to this, we enrich the plan model by associating each action with a set of *execution modalities*. While the action represents a possible way for achieving the high-level, qualitative effects, an execution modality refines this model to specify a particular configuration of rover's devices used to perform the action. Thus, the modalities of a given action have a significant impact on time and resources consumptions with significant variations from one another. While an initial assignment of modalities for the actions is decided on Earth, ReCon is in charge of adjusting such an assignment throughout the plan execution with the purpose of preserving the satisfaction of resources and time constraints.

After motivating the approach (Section 2), we formalize the notion of execution modalities (Section 3) upon which we show how this problem can be formulated as a Constraint Satisfaction Problem (CSP) and hence how

a CSP solver can be exploited for implementing ReCon (Section 4); it follows an exemplification of how ReCon actually works in a simple exploration mission (Section 5). Our preliminary tests (Section 6) show that ReCon is able to adjust on-line the mission configuration, actually preventing many plan failures.

2. MOTIVATIONS

In [MST12] we have presented a reactive supervisor, called ActS, that is in charge of monitoring the execution of a given mission plan: it has to detect anomalous trends that may occur during the execution of durative actions, and then it has to intervene by correcting, if possible, the rover's behavior.

The basic idea of ActS is that the failure of the (durative) action currently in progress can be prevented by properly adjusting the setting of some relevant parameters while the action is still in execution.

In [MST12] we have shown that, to accomplish its job, ActS needs in input an augmented model of the mission plan based on the PDDL 2.1 ([FL03]) formalism (it allows to explicitly model durative actions and invariant conditions, to capture constraints which must be satisfied during the whole action duration). The concept of durative actions, however, is not sufficiently expressive to model all the pieces of information required by ActS; for this reason we have augmented the plan model by associating each PDDL action with a set of temporal trends (both nominal and abnormal) that ActS has to recognize during the execution. Whenever an anomalous trend is identified, ActS intervenes by adjusting an appropriate set of parameters for correcting the current action execution.

The main advantage of ActS is its ability of detecting situations that may cause an action failure and of overcoming the critical situations without the need of stopping the plan execution. In many cases it is possible to restore a nominal trend of execution by adjusting the rover's parameter configuration, while the current action is still in progress. Of course, in situations where the parameters adjustment is not sufficient for correcting the anomalous trend, ActS interrupts the plan execution and invokes a mission replanner.

ActS is focused on the achievement of the expected effects of the current action. It is mainly concerned with tuning the parameters of the rover's devices in order not to violate the action invariant conditions; these conditions in fact represent bounds not to be passed for safety reasons. Thus ActS has the role of a short-term (reactive) supervisor, and it cannot estimate how its local decisions will impact the rest of the mission plan.

A mission rover example.

Let us assume that rover `r1` has to move from its initial location `l1` to reach a location `l2`, take a picture of such a target, and then move to location `l3` from which it can upload the collected data to Earth via a communication action. Such a simple mission is sketched in Figure 1. Let us assume moreover that the mission represents a feasible solution for a planning problem with goal: `{in(r1,l3), mem>=120, pwr>=0, time<=115}`, i.e.: at the end of plan the rover must be located in `l3` (propositional fluent), the free memory must be (at least) 120 Mbits, there must be a positive amount of power, and the mission must be completed within 115 secs.

The figure shows how the four actions (regular boxes) change the status of the rover over time (rounded-corner boxes). To simplify the picture, the rover's status is represented by just a subset of the whole status variables. Note that the status of a rover involves both propositional fluents; (e.g., `in(r1, l1)` meaning rover `r1` is in location `l1`), and numeric fluents; in particular, `memory` represents the amount of free memory (in Mbits) at a given step; `power` is the power (in Joule) available; `time` is the mission time given in seconds; finally, `com_cost` is an overall cost for communicating.

The picture (1) shows how the rover's status evolves when no unexpected situations occur. In particular, the estimates about the rover's status are inferred by predicting a deterministic set of effects. Let us now assume that during the execution of the first drive action the rover has to travel across a rough terrain. For safety reasons, ActS decides to reduce the speed, and as a consequence the drive action will take a longer time to be completed. While ActS's local decision is essential to avoid the failure of the drive action, it is easy to see that it causes the delayed execution of all the following actions. Consequently, one of the constraints associated with the goal (i.e., `time <= 115`) will be no longer satisfied if the forthcoming actions will be performed with their default settings.

It is worth noting that the mission goal could be missed even if ActS does not intervene. This may happen when the execution of the actions is nominal, in the sense that their expected high level effects are achieved, but the actual resource and time consumption is different from what was estimated at planning time. For instance, a drive action could take longer to be completed, not because ActS has slowed down the rover, but because the path was longer than estimated.

In order to reach the mission goal despite the uncertainty on the resource profiles at execution time, in this paper we propose to complement ActS with another agent, called ReCon. ReCon is a *long-term* supervisor, which considers all the actions still to be performed. It has two main goals: first, it verifies whether the current execution is still consistent with plan constraints, and hence the goal can be satisfied; second, in case the current execution would lead to the violation of any constraint, ReCon adjusts (when possible) the rest of the plan in order to restore the validity of the constraints on resources and time.

In the next section we will introduce the notion of *action*

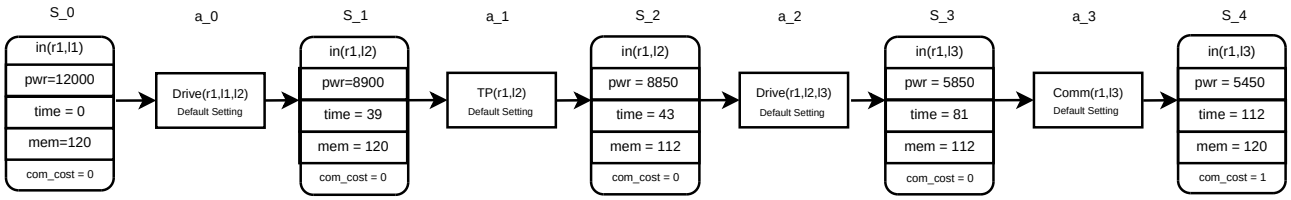

Figure 1. A simple mission plan.

modality and we will discuss how relying on this notion, ReCon is able to complement ActS's local decisions.

3. ACTION MODALITIES

We propose to augment the mission plan by associating each action instance with a set of alternative *modalities*. The intuition is that while actions differ each other in terms of qualitative effects, the expected result of an action can actually be obtained in many different ways by appropriately configuring the rover's devices. Of course, different configurations have in general different resource profiles and it is therefore possible that the execution of an action in a given configuration would lead to a constraint violation, whereas the same action performed in another configuration would not. We call these alternative configurations *modalities* and we propose to capture the impact of a specific modality by modeling the use of specific configurations in terms of pre/post conditions on resource and time profile, which become explicit in the action model definition.
Thanks to action modalities, ReCon has a chance to adapt the execution of the mission plan. Its job consists in selecting a modality for each action the rover has still to perform, such that the execution of these actions in their assigned modalities does not violate mission constraints.
In the rest of this section we formally introduce the notion of action modality and give an example; while in the following section we discuss how ReCon infers a new configuration of modalities.

Definition 1 Action model. *Let a be an action instance in the mission plan P, the model of a is represented by the tuple $\langle prop\_pre, prop\_eff, Mods\rangle$ where:*

- *$prop\_pre$ is the propositional preconditions set, which represents the applicability conditions for the execution of a*
- *$prop\_eff$ is the set propositional effects of a, i.e. a set of propositional atoms which hold after the action execution. Only if such atoms will be achieved the action itself is considered completed with success.*
- *$mods$ is a set of mutually exclusive modalities; each modality $m \in Mods$ corresponds to a particular setting S_m over the rover's parameters. m specifies how the resources change by executing a with modality m. Formally, m is a pair $\langle num\_pre, num\_eff\rangle$ where:*
  - *num_pre is a conjunction of comparison constraints among numeric expression (e.g., pwr_consumption < 3). As propositional preconditions, these constraints have to be satisfied in order to consider applicable the action in that modality.*
  - *num_eff specifies how the numeric fluents are changed by the action execution in that modality. For each numeric fluent the relation is either an increaser or a decreaser or an assigner (see an example below).*

The action model we propose splits the action behavior into two parts: one propositional at the higher level of abstraction and one numeric at the low level. The propositional part models the applicability conditions and the expected effects of action a. The modalities definition allows to specify how the action impacts the resource and time profile according to a specific modality.

Extending the mission rover plan model. By recalling the example of section 2, Figure 2 shows how we express a drive action by using the augmented model in a PDDL-like language. The action `drive (r1, l1, l2)` requires rover `r1` to move from location `l1` to location `l2`. `:modalities` introduces the set of modalities associated with a `drive`; in particular, we express for this action, three alternative modalities:
- `safe`: the rover moves slowly and far from obstacles;
- `normal`: the rover moves at its cruise speed and can go closer to obstacles;
- `agile`: the rover moves faster than `normal`.
The `:precondition` and `:effect` fields list as usual the applicability conditions and the expected effects, respectively, and are structured as follows: first a propositional formula encodes the precondition/effect of the action, then for each modality listed in `:modalities` a numeric expression is specified; such a numeric expression is used to model the amount of resources and time required (precondition) or consumed (effect) by the action when performed under that specific modality. The numerical part of the model (Mods) of the action is hence distributed in the preconditions and in the effects. For instance, in the preconditions `(reachable l1, l2)` and `(in r1, l1)` are two propositional atoms required as preconditions of the action. These two atoms

```
(:action drive
  :parameters (r1 l1 l2)
  :modalities (safe, normal, fast)
  :precondition (and (reachable l1 l2)(in r1 l1))
      (safe: ((>= (power r1) (f_pwr_safe(l1, l2)))
      (normal: ((>= (power r1) (f_pwr_normal(l1,l2)))
      (agile: ((>= (power r1) (f_pwr_agile(l1, l2))) )
  :effect (and
      (in r1 l2)
      (not(in r1 l1)))
      (safe:
         (decrease (power r1)(f_pwr_safe(l1, l2)))
         (increase (time) (f_time_safe(l1, l2))) )
      (normal:
         (decrease (power r1)(f_pwr_normal(l1, l2)))
         (increase (time) (f_time_normal(l1, l2))))
      (agile:
         (decrease (power r1)(f_pwr_agile(l1, l2)))
         (increase (time) (f_time_agile(l1, l2))) )
  ))
```

Figure 2. The augmented model of a `drive` *action.*

must be satisfied independently of the modality actually used to perform the `drive` action. While the expression `(safe -> (>= (power r1) (f_pwr_safe(l1,l2))))` means that the modality `safe` can be selected when the rover's power is at least greater than a threshold given by the function `(f_pwr_safe(l1,l2)))`, which estimates the power required to reach `l2` from `l1` in `safe` modality. Analogously, `(safe->(decrease(power r1)(f_pwr_safe(l1, l2)))` describes in the effects how the rover's power is reduced after the execution of the drive action. More precisely, we have modeled the power consumption as a function depending on the duration of the drive action (computed considering distance and speed) and the average power consumption per time unit given a specific modality. For instance, in safe modality, the amount of power consumed is estimated by the function `(f_pwr_safe(l1, l2))`, corresponding to `(safe_cons * (dist(l1,l2)/safe_speed)`, where `safe_cons` and `safe_speed` are the average consumption and the average speed for the safe modality, respectively, while `dist(l1,l2)` is the distance between the two locations `l1` and `l2`.
Finally, note that in the numeric effects of each modality, the model updates also the fluent `time` according to the selected modality. Also in this case, the duration the action is estimated by a function associated with each possible action modality.
Analogously to the drive action we model modalities also for the Take Picture (TP) and the Communication. For TP we have the low (LR) and high (HR) resolution modalities, whereas for the Communication we assume to have two different channels of communication: CH1 with low overall `comm_cost` and low bandwidth, and CH2 with high overall `comm_cost` but high bandwidth.

4. RECON: ADAPTIVE PLAN EXECUTION

In this section we describe how the plan adaptation process is actually carried on by exploiting a Constraint Satisfaction Problem representation. More precisely, three main algorithms drive the execution of ReCon: the global strategy specifying the control loop, and two important steps of this loop: **Update** by means of which new observations are asserted within the CSP representation and inconsistencies are discovered, and **Adapt** which has to adapt the plan.

4.1. The global strategy

Algorithm 1 shows the main steps of the adaptation process. The algorithm takes in input the initial rover's state S_0, the mission goal $Goal$, and the plan P enriched as discussed above. The algorithm returns $Success$ when the execution of the whole mission plan achieves the goal; $Failure$ otherwise. In this case, a failure means that there is no way to adapt the current plan in order to reach the goal satisfying mission constraints. To recover from this failure, a replanning step altering the structure of the plan should be invoked, but this step requires the intervention of the ground control station on Earth.

The first step of the ReCon algorithm is to build a $CSPModel$ representing the mission plan (line 1). Due to lack of space, we cannot present this step in details; our approach, however, is similar to the one presented by Lopex et al. in [LB03] in which the planning problem is addressed as a CSP.

However, while Lopex et al.'s work is mainly focused on propositional planning (and hence CSP variables are Boolean) in this work we are mainly focused on the satisfiability of numeric constraints. Thus, in creating the $CSPModel$, we associate each numeric fluent N^j of the original planning problem with a set N_i^j of numeric variables into the CSP representation, where $i : 0..|P| + 1$. The variable N_i^j will therefore denote the value of the numeric fluent N^j at the i-th step (i.e., before the application of the i-th action in the plan). The creation of the $CSPModel$ also includes the definition of constraints relating variables at step i with other variables at steps i+1 (see [LB03]). Within the $CSPModel$ there is another important set of variables: for each action $a_i \in P$, we create a variable mod_i whose domain is $Mods(a_i)$; variable mod_i therefore represents the modality assigned to action a_i. These modality variables are *decisional* variables in the CSP (i.e., variables that drive the search process), whereas the numeric fluent variables can be considered *non decisional* as they represent just the side effect of the modalities selection. Of course, at the time in which $CSPModel$ is created, each variable mod_i ($i : 1..|P|$) is initialized with the modality selected at planning time for a_i.

Once the $CSPModel$ has been built, the algorithm loops over the execution of the plan. Each iteration corresponds to the execution of the i-th action in the plan. First of all, ReCon submits action a_i to ActS (line 4), that is responsible for the on-line monitoring of the action execution. At the end of the action execution, ActS returns to Re-

Con a pair ($outcome$, obs_{i+1}); $outcome$ is either *Success* (when the propositional effects of the action have been achieved) or *Failure* (when the execution of the action has been interrupted due to the violation of some invariant conditions, or when the expected propositional effects have not been achieved); obs_{i+1} is the set of observations got after the execution of action a_i and hence referring to the $i+1$-th step of execution. In case $outcome$ is *Failure*, ReCon stops the plan execution and returns a failure; i.e., a replanning procedure is required. Otherwise, when $outcome$ is *Success*, ReCon exploits obs_{i+1} to update the $CSPModel$ (line 8). By propagating the observations inside the CSP representation, it is possible to verify the violation of some global constraints; whenever an inconsistency is found (line 10), ReCon has to adapt the current plan by finding an alternative assignments to action modalities that satisfies the numeric constraints (line 11). If the adaptation has success, a new non-empty plan $newP$ is returned and substituted to the old one. Otherwise, the plan cannot be adapted and a failure is returned; in this case, the plan execution is stopped and a new planning phase is needed starting from the actual state resulting after the execution of action a_i.

Algorithm 1: **ReCon**

```
   Input: S_0, Goal, P
   Output: Success or Failure
1  CSPModel = Init(S_0, Goal, P) ;
2  i = 0;
3  while ¬ P is completed do
4      (outcome, obs_{i+1}) = ActS(a_i, curMod(P, a_i));
5      if outcome is Failure then
6          return Failure
7      end
8      else
9          Update(CSPModel,a_i,num(obs_{i+1}));
10         if ¬ numeric-consistent(CSPModel,Goal)
           then
11             newP = Adapt(CSPModel,i,Goal,P);
12             if newP ≠ ∅ then
13                 P = newP
14             end
15             else
16                 return Failure
17             end
18         end
19     end
20 end
21 return Success
```

4.2. Update

The **Update** step is sketched in Algorithm 2. The algorithm takes in input the CSP model to update, the last performed action a_i, and the set $NObs$ of observations about numeric fluents. The algorithm starts by asserting within the model that the i-th action has been performed; see lines 1 and 2 in which variable mod_i is constrained to assume the special value $exec$. In particular, a first role of the $exec$ value is to prevent the adaptation process to change the modality of an action that has already been performed, as we will see in the following section. Moreover, $exec$ allows also the acquisition of observations even when the observed values are completely unexpected. In fact, by assigning the modality of action a_i to $exec$, we relax all the constraints over the numeric variables at step $i+1$-th (which encode the action effects). This is done in lines 3-6 in which we iterate over the numeric fluents N^j mentioned in the effects of action a_i, and assign to the corresponding variable at $i+1$-th step the value observed in $NObs$. On the other hand, all the numeric fluent that are not mentioned in the effects of action a_i do not change, so the corresponding variables at step $i+1$ assume the same values as in the previous i-th step (lines 7-10).

Algorithm 2: **Update**

```
   Input: CSPModel, a_i,NObs
   Output: modified CSPModel
1  removeConstraint(CSPModel,mod_i=curMod(a_i));
2  addConstraint(CSPModel,mod_i=exec);
3  foreach N^j ∈ affected(a_i) do
4      addConstraint(CSPModel,
5          (mod_i=exec)→ N^j_{i+1}=get(NObs,N^j_{i+1}))
6  end
7  foreach N^j ∈ ¬affected(a_i) do
8      addConstraint(CSPModel,
9          (mod_i=exec)→ N^j_{i+1}=N^j_i)
10 end
```

4.3. Adapt

The **Adapt** module, shown in Algorithm 3, takes in input the CSP model, the index i of the last action performed by the rover, the mission goal, and the plan P; the algorithm returns a new adapted plan, if it exists, or an empty plan when no solution exists.

The algorithm starts by removing from $CSPModel$ the constraints on the modalities of actions still to be performed; i.e., each variable mod_k with k greater than i is no longer constrained (a_i is the last performed action and its modality is set to $exec$) (lines 1-4). This step is essential since the current $CSPModel$ is inconsistent; that is, the current assignment of modalities does not satisfies the global constraints. By removing these constraints, we allow the CSP solver to search in space of possible assignments to modality variables (i.e., the decisional ones), and find an alternative assignment that satisfies the global constraints (line 5). If the solver returns an empty solution, then there is no way to adapt the current plan and **Adapt** returns no solution. Otherwise (lines 9-16), at least a solution has been found. In this last case, a new assignment of modalities to the variables mod_k ($k : i+1..|P|$) is extracted from the solution, and this assignment is returned to the ReCon algorithm

as a new plan $newP$ such that the actions are the same as in P, but the modality labels associated with the actions $a_{i+1}, .., a_{|P|}$ are different.

Note that, in order to keep updated the CSP model for future adaptations, the returned assignment of modalities is also asserted in $CSPModel$; see lines 11-14.

Algorithm 3: **Adapt**

```
   Input: CSPModel, i,Goal,P
   Output: a new plan, if any
 1 for k=i+1 to |P| do
 2 |   removeConstraint(CSPModel,
 3 |             mod_k=currentMod(a_k));
 4 end
 5 Solution = solve(CSPModel);
 6 if Solution = null then
 7 |   return ∅
 8 end
 9 else
10 |   newP=extractModalitiesVar(Solution);
11 |   for k=i+1 to |newP| do
12 |   |   addConstraint(CSPModel,
13 |   |     mod_i=curMod(newP[i]));
14 |   end
15 |   return newP
16 end
```

5. RUNNING THE MISSION ROVER EXAMPLE

Let us consider again the example in Figure 1, and let us see how RoCon manages its execution. First of all, the plan model must be enriched with the execution modalities as explained in Section 3; Figure 3 (top) shows the initial configuration of action modalities: the drive actions have `cruise` modalities, the take picture (TP) has `LR` (low resolution) modality, and the communication (`Comm`) uses the low bandwidth channel (`CH1`). This is the enriched plan ReCon receives in input.

Now, let us assume that the actual execution of the first drive action takes a longer time than expected, 47s instead of 38s, and consumes more power, 3775 Joule instead of 3100 Joule. While the discrepancy on power is not a big issue as it will not cause a failure, the discrepancy on time will cause the violation of the constraint `time <=115`; in fact, performing the subsequent actions in their initial modalities would require 120 seconds. In other words, the assignment of modalities to the subsequent actions does not satisfies the mission constraints. This situation is detected by ReCon that intervenes and, by means of the **Adapt** algorithm discussed above, tries to find an alternative configuration of modalities.

Let us assume that communication cost is constrained; that is, the mission goal includes the constraint `com_cost = 1`; this prevents ReCon from using the fast communication channel. In order to gain some time, ReCon can just switch the modality of the second drive action from `cruise` to `agile`, which allows the rover to move faster. Figure 3 (bottom), shows the reconfigured plan with the estimation on the resource and time consumption. Note that the new configuration has an impact not only on `time`, but also on `power`; such an impact, however, does not represent a violation of the global constraints (the constraint on power is not so strict), and hence the proposed (re)configuration is a solution.

6. EXPERIMENTAL VALIDATION

The approach described in the paper aims at detecting deviations from the expected use of resources, and in reconfiguring the forthcoming actions by selecting alternative modalities once relevant deviations can prevent the plan accomplishment. The reconfiguration purpose is intended to accommodate the actual plan execution in such a way that all the constraints continue to be satisfied.

Therefore it is quite obvious that the approach is of limited use in case there is no discrepancy between the predicted and actual consumption of the resources. For this reason we have designed an experimental setting which allows us to evaluate the performance of our system depending on the degree of discrepancy encountered throughout the execution of the plan.

We have designed 170 test cases, each of which involves a plan composed of 17-18 activities (typically 10-11 drives, 6-7 take pictures, 1-2 communications). The initial plan is fully instantiated (a modality has been assigned to each action) and is feasible since the constraints built upon all the propositional and numerical aspects defined for the problem at hand are satisfied. The 170 test cases have been subdivided into two subsets: 102 *hard* cases and 68 *medium* cases. The classification depends on the degree of strictness we have imposed on the numeric constraints: in the *hard* cases numeric constraints on the resources are very strict since they are just sufficient for completing the plan and there is a very little tolerance for accommodating deviations. In the *medium* cases the constraints are strict but there is a larger margin for tolerating deviations.

ReCon has been implemented in Java 1.7, the Choco CSP solver (version 2.1.3) [1] has been used in the **Adapt** algorithm to find an alternative configuration. Each case has been run in ten different scenarios [2]: in scenario 1 the observations coming from the simulated execution coincide with the predicted values, in scenario 10 the discrepancy between observed values and predicted ones is high (over 32%), in the intermediate scenarios there is an increasing amount of discrepancy between the two extremes values (0 and 32%). In order to (i) simulate exogenous events and (ii) have a comprehensive experimental setting, to each action we associate an ordered set of noised "real

[1] http://www.emn.fr/z-info/choco-solver/

[2] Experiments have run on a 2.53GHz Intel(R) Core(TM)2 Duo processor with 4 GB.

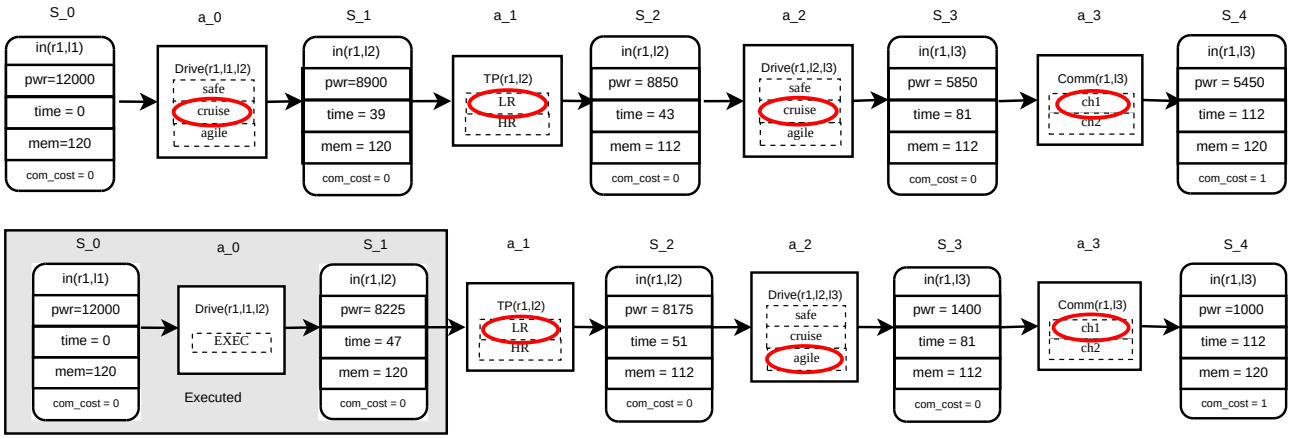

Figure 3. The initial configuration of modalities (above), and the reconfigured plan (below).

action": the set is ordered wrt the amount of noise we considered. Therefore, for each i-th scenario, the execution of action A is simulated by picking up the i-th real action B associated to A and hence the status is transformed by taking into account effects of action B instead of A.

We compared the performance of the proposed system involving ReCon with a basic architecture which excludes it but performs the consistency check. In case a constraint violation is detected, the basic architecture stops with a failure while the proposed system looks for a new plan configuration. We measured the performance of the two systems as the amount of the plan which has been executed before a failure (let us call it PPE). Ideally, PPE is 100% when the system is able to complete the plan with success; the architecture with ReCon may of course include the execution of pieces of plan which have been subject of reconfigurations.

Figure 4 and 5 report the data concerning the performance for the *medium* cases and the *hard* cases, respectively. Along the abscissa we have the identifier of the scenario (from 1 to 10), whereas the ordinate values correspond to the percentage of plan executed with success. For the medium cases, it is easy to see that for small degrees of deviations the average performance is quite close to 100% for both architectures, while they show different degrees of ability in carrying on the plan when the degree of discrepancy increases. It is worth noting that the reconfiguration system is able to deal effectively also with high degree of deviations (PPE over 70% for the 10-th scenario where the discrepancy is high).

When we consider *hard* cases the impact of the reconfiguration module is even more apparent. The basic system has a limited PPE even when the degree of discrepancy is small since the resource constraints are very strict and the possibility of occurrence of a failure is high, while ReCon is able to achieve high levels of PPE even for large values of discrepancy. Of course, this improvement of performance has a computational cost due at the time spent by the CSP solver to find a consistent modalities assignment.

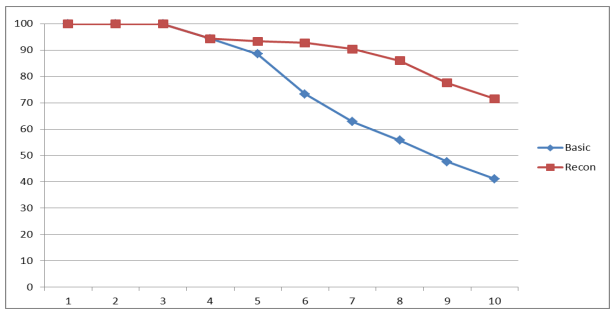

Figure 4. Percentage of Plan Completion for the medium difficulty cases

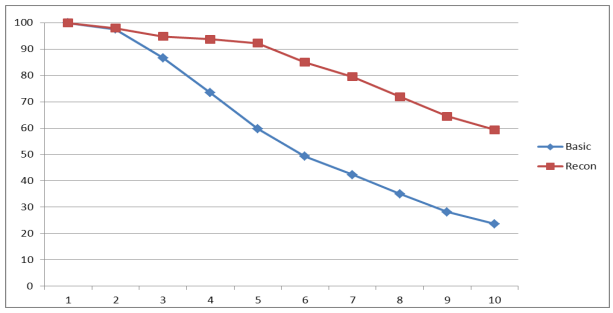

Figure 5. Percentage of Plan Completion for the Hard Difficulty Cases

However, by observing Table 1, the average CPU time for each reconfiguration in most cases is negligible and even in the worst case is acceptable. Figure 6 shows the average number of reconfigurations for each scenario; the experiment results confirm that the activation of ReCon is more frequent as the discrepancy increases. It is worth noting that for hard cases the CPU time increases since the Choco solver has to perform much more search when the problem is severely constrained.

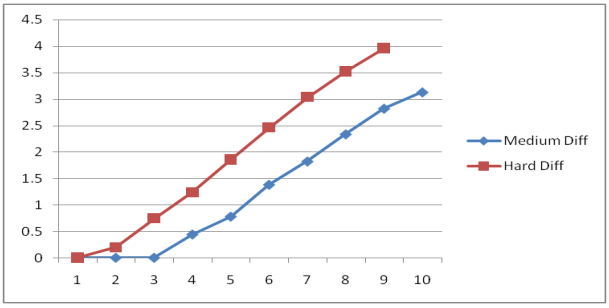

Figure 6. Average Number of Reconfiguration

	Medium	Hard
Average Recon Cpu Time (msec)	119 (±11)	244 (±44)
Max Recon Cpu Time (msec)	1412	8983

Table 1. Cpu Time

7. CONCLUSIONS

While many approaches to robust plan execution [GS10, vdKdW05, GGO10] are based on a re-planning step to recover from plan failures, in this paper we have proposed an alternative methodology based on the re-configuration of the mission. Our approach leaves the high-level structure of the plan (i.e., the sequence of mission tasks) unchanged, but operates at the level of *modalities* associated with each action in the plan. This idea stems by the observation that, albeit a planetary rover carrying on a space exploration mission has to exhibit some form of autonomy, its autonomy is often bounded by two main factors: (1) the on-board computational power is not always sufficient to handle mission recovery problems, and (2) the rover cannot in general deviate from the given plan without the approval from the ground control station.

The strategy we propose has therefore the advantage of maintaining the structure of the mission plan unchanged while endowing the rover with an appropriate level of autonomy for handling unexpected contingencies. In this paper we have presented a module, ReCon, in charge of finding a configuration of modalities that does not violate the global constraints on the use of resources included in the mission goal. As we have discussed, ReCon is located in between ActS (focused on the execution of a single action) [MST12] and a mission (re)planner (on Earth). The idea is that the intervention of ReCon can reduce the number of expensive invocations to the mission (re)planner. The experimental results we have presented, show in fact that the percentage of mission accomplished by the rover is significantly higher when ReCon is active, than when it is inactive in the basic approach. The advantages of ReCon are evident even in the *hard* cases, where high degrees of discrepancies and strong constraints make the plan execution particularly challenging.

As we have discussed above, the implementation of ReCon exploits a generic CSP solver to find a consistent assignment of modality; in our specific case we have adopted the Choco solver, which was the fastest CSP solver implemented in Java in the Fourth International Constraint Solver Competition (CSC'09) [3]. Since in Choco many features involving real-valued variables are still under development, in our current implementation we had to map real-valued variables into appropriate integer variables (without losing too much precision).

The approach we have presented can be improved in a number of ways. A first important improvement is the search for an optimal solution. In the current version, in fact, ReCon just finds one possible configuration that satisfies the global constraints. In general, one could be interested in finding the best configuration that optimizes a given objective function. Reasonably, the objective function could take into account the number of changes to action modalities; for instance, in some cases it is desirable to change the configuration as little as possible. Moreover, the objective function could be defined to prefer configurations that adhere as close as possible to the resource profile that was estimated at planning time. Of course, the search for an optimal configuration is justified when the global constraints are not strict, and several alternative solutions are possible.

REFERENCES

[BWW06] Stephen A. Block, Andreas F. Wehowsky, and Brian C. Williams. Robust execution of contingent, temporally flexible plans. In *Workshop on Cognitive Robotics*, 2006.

[CSW09] P. R. Conrad, J. A. Shah, and B. C. Williams. Flexible execution of plans with choice. In *ICAPS'09*, pages 74–81, 2009.

[FL03] M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *JAIR*, 20:61–124, 2003.

[GGO10] A. Garrido, C. Guzman, and E. Onaindia. Anytime plan-adaptation for continuous planning. In *PlanSIG'10*, 2010.

[GS10] A. Gerevini and I. Serina. Efficient plan adaptation through replanning windows and heuristic goals. *Fundamenta Informaticae*, 102(3-4):287–323, 2010.

[LB03] A. Lopez and F. Bacchus. Generalizing graphplan by formulating planning as a csp. In *Proceedings of IJCAI'03*, pages 954–960, 2003.

[MST12] R. Micalizio, E. Scala, and P. Torasso. Towards robust execution of mission plans for planetary. *ACTA Futura*, 5, 2012.

[vdKdW05] R. van der Krogt and M. de Weerdt. Plan repair as an extension of planning. In *ICAPS'05*, pages 161–170, 2005.

[3] http://www.cril.univ-artois.fr/CPAI09/