ReCon: an Online Task ReConfiguration Approach for Robust Plan Execution

Enrico Scala*[1], Roberto Micalizio†[1], and Pietro Torasso‡[1]

[1]Dipartimento di Informatica - Universita' di Torino, Italy

January 30, 2016

Abstract

The paper presents an approach for the robust plan execution in presence of consumable and continuous resources. Plan execution is a critical activity since a number of unexpected situations could prevent the feasibility of tasks to be accomplished; however, many robotic scenarios (e.g. in space exploration) disallow robotic systems to perform significant deviations from the original plan formulation. In order to both (i) preserve the "stability" of the current plan and (ii) provide the system with a reasonable level of autonomy in handling unexpected situations, an innovative approach based on task reconfiguration is presented. Exploiting an enriched action formulation grounding on the notion of execution modalities, ReCon replaces the replanning mechanism with a novel reconfiguration mechanism, handled by means of a CSP solver. The paper studies the system for a typical planetary rover mission and provides a rich experimental analysis showing that, when the anomalies refer to unexpected resources consumption, the reconfiguration is not only more efficient but also more effective than a plan adaptation mechanism. The experiments are performed by evaluating the recovery performances depending on constraints on computational costs.

1 Introduction

The management of a plan for a robotic agent operating in hazardous and extreme environments is a critical activity that has to take into account several challenges. In particular, in the context of space exploration, a planetary rover operates in an environment which is just partially observable and loosely predictable. As a consequence, the rover must have some form of autonomy in order

*scala@di.unito.it

†micalizio@di.unito.it

‡torasso@di.unito.it

to guarantee robust plan execution (i.e., reacting to unexpected contingencies). The rover's autonomy, however, is typically bounded both because of limitations of on-board computational power, and because the rover is not in general allowed to change significantly the high level plan synthesized on Earth. Space missions therefore exemplify situations where contingencies occur, but plan repair must be achieved through novel techniques trading-off rover's autonomy and the stability of the mission plan.

Robust plan execution has been tackled in two ways: on-line and off-line. On-line approaches, such as [14, 18, 13, 4, 27, 20], interleave execution and replanning: whenever unexpected contingencies 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. Off-line approaches, such as [3, 8], avoid replanning by anticipating, at planning time, the possible contingencies. The result of such a planning phase is a contingent plan that encodes choices between functionally equivalent sub-plans[1]. At execution time, the plan executor is able to select a contingent plan according to the current contextual conditions. However, as for instance in the work of [24], the focus is mainly on the temporal dimension and they do not consider consumable and continuous resources.

In this paper we propose a novel on-line methodology to achieve robust plan execution, which is explicitly devised to deal with unexpected deviations in the consumption of rover's resources. First, in line with the action-based approach *a-la* STRIPS [11] and differently from the constrained based planning [12, 22], we model consumable resources as numeric fluents (introduced in PDDL 2.1 [11]). Then, we enrich the model of the rover's actions by introducing a set of *execution modalities*. The basic idea is that the propositional effects of an action can be achieved under different configurations of the rover's devices. These configurations, however, may have a different impact on the consumption of the resources. An *execution modality* explicitly models the resource consumption profile when an action is carried out in a given rover's configuration. The integration of *execution modality* at the PDDL level allows a seamless integration between planning and execution.

Relying on the concept of execution modalities, we propose to handle exceptions arising in planetary rover domains as a reconfiguration of action modalities, rather than as a replanning problem. In particular, the paper proposes a plan execution strategy, denoted as ReCon; once (significant) deviations from the nominal trajectory are detected, ReCon intervenes by reconfiguring the modalities of the actions still to be performed with the purpose of restoring the validity of resource constraints imposed by the rover mission.

To accomplish its task ReCon uses Choco[2] as CSP solver, so that it takes advantage of both the power of the constraint programming and the high level representation of PDDL.

After introducing a motivating example, we describe the employed action

[1]The notion of alternative (sub)plans is also presented for (off-line) scheduling; for details see [1].

[2]The software is at disposal at http://www.emn.fr/z-info/choco-solver/, while the work has been presented in [23].

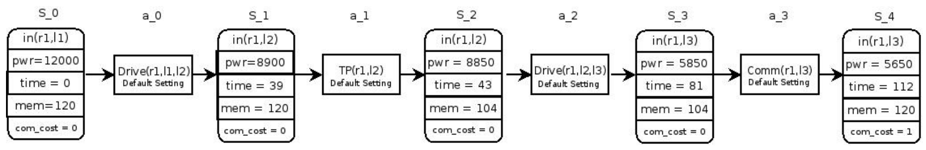

Figure 1: A simple mission plan.

model, enriched with the notion of execution modality. Then we introduce the ReCon strategy and an example showing how the system actually works in a exploration rover mission. Finally, an experimental section, which evaluates the competence and the efficiency of the strategy w.r.t. a traditional replanning from scratch and the LPG-ADAPT system reported in [15].

2 Motivating Example

Let us consider a planetary rover in charge of exploring (and analyzing) a number of potentially interesting sites and able to transmit information towards the Earth. In doing so the rover is capable of moving, taking pictures, and starting the data upload once the pieces of information must be transmitted. For simplicity reasons, consider the mission plan of Figure 1, involving *take picture, drive* and *communications* activities. This mission represents a feasible solution for a planning problem with goal: `{in(r1,l3), mem>=120, pwr>=0, time<=115}` ; that is, at the end of plan the rover must be located in `l3` (propositional fluent), the free memory must be (at least) 120 memory units, 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 the time (rounded-corner boxes)[3]. 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: `memory` represents the amount of free memory, `power` is the amount of available power, `time` is the mission time given in seconds, and `com_cost` is an overall cost associated with communications.

The estimates about the rover's status are inferred by predicting, deterministically, the effects of the actions. In particular, the numeric fluents have been estimated by using a "default setting" (i.e., a standard modality) associated with each action.

Let us now assume that during the execution of the first drive action the rover has to travel across a rough terrain. Such an unexpected condition affects the drive as the rover is forced to slowdown[4], and as a consequence the drive action takes a longer time to be completed; the effects are propagated till the last snapshot, *s_4* where the goal constraint `time <= 115` is no longer satisfied.

[3]To simplify the picture, we show in the rover's status just a subset of the whole status variables.

[4]The slowdown command of the rover may be the consequence of a reactive supervisor, which operates as a continuous controller as shown in [21].

After detecting this inconsistency, approaches based on a pure replanning step would compute a new plan achieving the goal by changing the original mission. For instance, some actions could be skipped in order to compensate the time lost during the first drive.

However, robotic systems as a planetary rover have typically different configurations of actions to be executed and each configuration can have a different impact on the mission progress. For instance the robotic systems described in [5] and in [21] can perform a drive action in fast or slow modes. Reliable transmission to the earth, for example, can be slow and cheap, or fast and expensive, depending on the devices actually used.

Our proposal is to explicitly represent such different configurations within the action models, and hence try to resolve an impasse via a reconfiguration of the actions still to be performed. Intuitively, our objective is to keep the high level plan structure unchanged, but to adjust the modalities of the actions still to be performed. In section 5 we will see an example of such a reconfiguration.

In the next section we will introduce the rover action model that explicitly expresses the set of *execution modality* at disposal.

3 Modeling Rover's Actions

As shown in the previous section, a planetary rover can perform the same set of actions via different configurations of parameters or devices. To capture this aspect, this section introduces the rover action model adopted in this work. As reported in [25], the model exploits (and extends) the numeric PDDL 2.1 action model [11], i.e. where the numeric fluent notion has been proposed. In particular, we use the numeric fluents to model continuous and consumable resources.

The intuition is that, while actions differ each other in terms of qualitative effects (e.g. a drive action models how the position of the rover changes after the action application), the expected result of an action can actually be obtained in many different ways by appropriately configuring the rover's devices (e.g. the drive action can be performed with several engine configurations). 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 the numeric fluents involved; such *modalities* become explicit in the action model definition.

The resulting model expresses the rover actions at two different levels of abstraction. The higher one is the qualitative level indicating "what" the action does. The lower one is the quantitative level expressing "how" the action achieves its effect.

The idea of *alternative behaviors* has also been investigated in (off-line) scheduling, where the notion of Temporal Network with Alternatives has been introduced [1]. It is quite evident however that, as anticipated in the introduction, the concept of execution modality is inspired to an (on-line) action centered approach [4], rather than on a constraints/scheduling based one [6].

By recalling our motivating example, Figure 2 shows the model of the drive action. The action template `drive (?r, ?l1, ?l2)` requires a rover `?r` to move from a 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; intuitively the action should spend more time but consuming less power
- `cruise`: the rover moves at its cruise speed and can go closer to obstacles;
- `agile`: the rover moves faster than `cruise`, consuming more power but requiring less time.

The `:precondition` and `:effect` fields list the applicability conditions and the effects, respectively, and are structured as follows: first a propositional formula encodes the condition under which the action can be applied; the second field (`:effect`) indicates the positive and the negative effects of the action. For each modality m in `:modalities` we have the amount of resources required (numeric precondition) or consumed/produced (numeric effect) by the action when performed under that specific modality m.

For instance, the preconditions `(reachable ?l1, ?l2)` and `(in ?r1, ?l1)` are two atoms required as preconditions for the application of the action. These two atoms must be satisfied independently of the modality actually used to perform the `drive` action. While the comparison `(safe: (>= (power ?r) (* (safe_cons ?r) (/ (distance ?l1 ?l2) (safe_speed ?r)))))` means that the modality `safe` can be selected when the rover's power is at least larger than a threshold given by evaluating the expression on the right side. Analogously,
`(safe: (decrease (power ?r) (*(safe_cons ?r) (/ (distance ?l1 ?l2) (safe_speed ?r))))` 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 depends on two parameters `(safe_cons ?r)` and `(safe_speed ?r)` which are the average consumption and the average speed for the safe modality, respectively, while `(distance ?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 of 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 (COMM). For TP we have the low (LR) and high (HR) resolution modalities which differ in the quality of the taken picture and the occupied memory. Intuitively, the more the resolution is, the more the memory consumption will be.

```
(:action drive
  :parameters ( ?r - robot ?l1 - site ?l2 - site)
  :modalities (safe,normal,agile)
  :precondition (and (in ?r ?l1) (road ?l1 ?l2)
  (safe: (>= (power ?r) (* (safe_cons ?r)
     (/ (distance ?l1 ?l2) (safe_speed ?r)))))
  (cruise: (>= (power ?r) (* (cruise_cons ?r)
             (/ (distance ?l1 ?l2) (cruise_speed ?r)))))
  (agile: (>= (power ?r) (* (agile_cons ?r)
             (/ (distance ?l1 ?l2) (agile_speed ?r)))))
  )
  :effect
  (and
    (in ?r ?l2) (not (in ?r ?l1))
    (safe: (decrease  (power ?r) (* (safe_cons ?r)
            (/ (distance ?l1 ?l2) (safe_speed ?r))))
           (increase  (time) (/ (distance ?l1 ?l2)) (safe_speed ?r)))
           (increase  (powerC ?r) (* (safe_cons ?r)
                           (/ (distance ?l1 ?l2) (safe_speed ?r))))
    (cruise: (decrease  (power ?r) (* (cruise_cons ?r)
                           (/ (distance ?l1 ?l2) (cruise_speed ?r))))
           (increase  (time) (/ (distance ?l1 ?l2)) (cruise_speed ?r))
           (increase  (powerC ?r) (* (cruise_cons ?r)
                           (/ (distance ?l1 ?l2) (cruise_speed ?r)))))
    (agile: (decrease  (power ?r) (* (agile_cons ?r)
                           (/ (distance ?l1 ?l2) (agile_speed ?r))))
           (increase  (time) (/ (distance ?l1 ?l2)) (agile_speed ?r))
           (increase  (powerC ?r) (* (agile_cons ?r)
                           (/ (distance ?l1 ?l2) (agile_speed ?r)))))
  )
```

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

Figure 3 reports the model of the communication action; it is worth noticing that execution modalities correspond to two different communication channels: CH1 with low overall `comm_cost` and low bandwidth, and CH2 with high overall `comm_cost` but high bandwidth.

The selection of action modalities has to take into account that complex dependencies among resources could exist. For instance, even if a high resolution TP takes the same time as a low resolution TP, the selection has a big impact on the amount of time spent globally, too. As a matter of facts, as long as the amount of stored information increases, the time spent by a (possible) successive COMM grows up accordingly, which means that also the global mission horizon will be revised.

Given the rover's actions defined so far, a rover mission plan is a total ordered set of fully instantiated rover's action templates[5]. Given a particular

[5]The plan can be also generated automatically by exploiting a numeric planner system,

```
(:action comm
  :parameters ( ?r - robot ?l1 - site )
 :modalities (ch1,ch2)
  :precondition (and(in ?r ?l1)

 (ch1:   (and (> (memoryC ?r) 0) ( >= (power ?r)
              (/ (memoryC ?r) (bandwith-ch1 ?r)))))
 (ch2:   (and (> (memoryC ?r) 0) ( >= (power ?r)
              (/ (memoryC ?r) (bandwith-ch2 ?r)))))

  :effect
  (and (infoSent ?r ?l1)
     (ch1:   (assign (memoryC ?r) 0)
         (increase (memory ?r) (memoryC ?r))
             (increase (time) (/ (memoryC ?r) (bandwith-ch1 ?r)))
             (increase (powerC ?r) (* (ch1-cons ?r)
                       (/ (memoryC ?r) (bandwith-ch1 ?r)))
             (decrease (power ?r)  (* (ch1-cons ?r)
                       (/ (memoryC ?r) (bandwith-ch1 ?r)))
             (increase (comm_cost) 1)))
     (ch2:   (assign (memoryC ?r) 0)
             (increase (memory ?r) (memoryC ?r))
             (increase (time) (/ (memoryC ?r) (bandwith-ch2 ?r)))
             (increase (powerC ?r) (* (ch2-cons ?r)
                       (/ (memoryC ?r) (bandwith-ch2 ?r))))
             (decrease (power ?r)  (* (ch2-cons ?r)
                   (/ (memoryC ?r) (bandwith-ch2 ?r))))
             (increase (comm_cost) 3))
)
```

Figure 3: The augmented model of a `communication` action.

rover's state S and a given set of goals G to be reached (including both propositional/classical conditions and constraints on the amount of resources), the mission plan is valid iff it achieves G from S.

Executing the mission plan. As we have seen in the previous section, the rover's mission can be threatened many times by unexpected contingencies; so the validity of the mission can be easily compromised during its actual execution.

Nevertheless, when the detected unexpected contingency at execution time just invalidates the resource consumption expectations, even if the current modality allocation would not be consistent with the constraints involved in the plan and in the goal, there could be "other" allocations of modalities still feasible. By exploiting this intuition, the next section introduces an adaptive execution technique which, instead of abandoning the mission being executed, tries first to repair the flaws via a reconfiguration of the action modalities. The

properly modified to handle actions with modalities (e.g., the Metric-FF planning system [17] or LPG [16]).

reconfiguration considers all those actions still to be executed.
Given a plan P, to indicate when a plan is just *resource inconsistent*, we will use the predicate *res_incon* over P, i.e. we will say *res_incon(P)*. Otherwise we will say that the plan is valid or structurally invalid. This latter case happens when, given the current plan formulation, at least an action in the plan is not propositional applicable, or there is at least a missing (propositional) goal.

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. The main strategy implemented, namely ReCon, is a continual planning agent [4],[9], extended to deal with the rover actions model presented in the previous section. In order to handle the CSP representation, ReCon exploits two further submodules: **Update** by means of which new observations are asserted within the CSP representation, and **Adapt** which has the task of making the mission execution adaptive to the incoming situation.

4.1 The Continual Planning Loop

Algorithm 1 shows the main steps required to execute and (just in case) adapt the plan being executed. The algorithm takes in input the initial rover's state S_0, the mission goal $Goal$, and the plan P expressed as discussed in the previous section. Note that each action has to have a particular modality of execution instantiated. 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 algorithm is to build a $CSPModel$ representing the mission plan (line 1). As thoroughly described in [25], our approach inherits the main steps by Lopez et al. in [19] in which the planning problem is addressed as a CSP[6]. As a difference w.r.t. the classical planning, the encoding exploited by our approach needs to store variables for the modalities to be chosen, and variables for the numeric fluents involved in the plan. Numeric fluents variables are replicated as many steps in the plan. The purpose is to capture all the possible evolutions of resources profiles given the modalities that will be selected. The constraints oblige the selection of the modality to be consistent with the resource belonging to the previous and successive time step. Moreover, further constraints allow only reconfigurations consistent with the current observation acquired (which at start-up corresponds to the initial state), and the goals/requirement of the mission.

[6]Alternative CSP conversions are possible; for instance see [2].

Algorithm 1: ReCon

```
Input: $S_0$, $Goal$, $P$
Output: $Success$ or $Failure$
1  $CSPModel = Init(S_0, Goal, P)$ ;
2  $i = 0$;
3  while $\neg$ $P$ is completed do
4      execute($a_i$, $curMod(a_i)$);
5      $obs_{i+1}$ = observe();
6      if $P$ is structurally invalid w.r.t. $obs_{i+1}$ and $Goal$ then
7          return $Failure$
8      else
9          Update($CSPModel$,$a_i$,$num(obs_{i+1})$);
10         if $res\_incon(P)$ then
11             $newP$ = Adapt($CSPModel$,$i$,$Goal$,$P$);
12             if $newP \neq \emptyset$ then
13                 $P = newP$
14             else
15                 return $Failure$
16     $i = i + 1$
17 return $Success$
```

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. At the end of the action execution the process verifies the current observation obs_{i+1} with the rest of the mission to be executed. In case the plan is structurally invalid (some propositional conditions are not satisfied or the goal cannot be reached) ReCon stops the plan execution and returns a failure; i.e., a replanning procedure is required.

Otherwise we can have two other situations. First, there have been no consistent deviations from the nominal predictions therefore the execution can proceed with the remaining part of the plan. Second the plan is just resource inconsistent ($res\_incon(P)$, line 10). In this latter case, 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. This new plan is actually the old plan, but with a different allocations of action modalities. 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.

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 3-5 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 fluents 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 6-8). The idea of the Update is to make the CSP aware of the current new observations and the modalities already executed. In this way, a reconfiguration task does not need to rebuild the structure completely from scratch.

Algorithm 2: Update

```
   Input: $CSPModel$, $a_i$,$NObs$
   Output: modified $CSPModel$
1  delConstraint($CSPModel$,$mod_i$=curMod($a_i$));
2  addConstraint($CSPModel$,$mod_i$=$exec$);
3  foreach $N^j \in affected(a_i)$ do
4      addConstraint($CSPModel$,
5        ($mod_i$=$exec$)$\rightarrow$ $N^j_{i+1}$=get($NObs$,$N^j_{i+1}$))
6  foreach $N^j \in \neg affected(a_i)$ do
7      addConstraint($CSPModel$,
8        ($mod_i$=$exec$)$\rightarrow$ $N^j_{i+1}$=$N^j_i$)
```

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 1-2). 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 the space of possible assignments to modality variables (i.e., the actual decisional variables, since the numeric fluents are just side effects of the modality selection), and find an alternative assignment that satisfies the global constraints (line 3). If the solver returns an empty solution, then there is no way to adapt the current plan and **Adapt** returns no solution. Otherwise (lines 6-10), 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 6 to 10.

Algorithm 3: Adapt

```
   Input: $CSPModel$, $i$,$Goal$,$P$
   Output: a new plan, if any
1  for $k=i+1$ to $|P|$ do
2      delConstraint($CSPModel$ $mod_k$=currentMod($a_k$))
3  $Solution$ = solve($CSPModel$);
4  if $Solution$ = $null$ then
5      return $\emptyset$
6  else
7      $newP$=extractModalitiesVar($Solution$);
8      for $k=i+1$ to $|newP|$ do
9          addConstraint($CSPModel$, $mod_i$=curMod($newP[i]$))
10     return $newP$
```

5 Running the Mission Rover Example

Let us consider again the example in Figure 1, and let us see how ReCon manages its execution. First of all, the plan model must be enriched with the execution modalities as previously explained; Figure 4 (top) shows the initial configuration of action modalities: the drive actions have `cruise` modalities, the take picture (TP) has `HR` (high 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

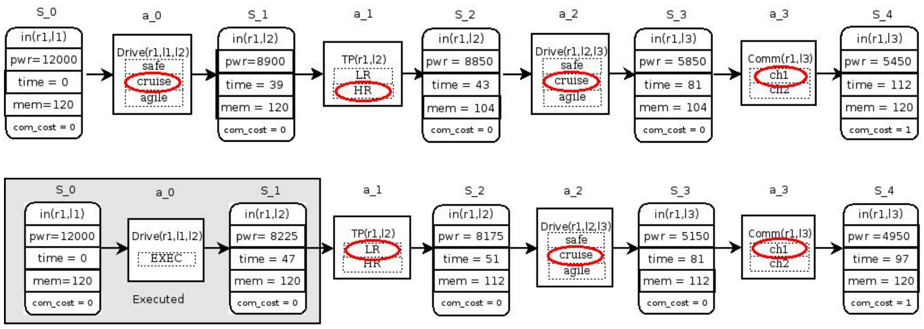

Figure 4: The initial configuration of modalities (above), and the reconfigured plan (below).

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. The more intuitive decision is to promote the execution of the *drive* to *agile*. However, this would cause the violation on the constraint concerning the maximum amount of power to be spent. Therefore ReCon has to look for an alternative assignments of modalities.

It is interesting to note that a lower resolution image consumes less memory, meaning that the successive communication, in our case *(COMM R1 L3)*, will need less time (and also less power) for achieving its effects. For this reason ReCon demotes the next activity, i.e. TP, to be execute to modality LR and so the global constraints are now satisfied.

Of course, we assume that mission constraints leave ReCon some room to repair resource inconsistent situations. For instance, if the mission has required an hard constraint on the quality of the taken images, the low resolution would have not been possible, and hence an overall replanning would have been necessary.

In principle, by flattening all the actions and the given modalities as explained in [28], replanning is possible as alternative to the reconfiguration mechanism. In this case, however, the problem to be handled would become much more difficult, since all the possible action sequences applicable starting from the current state could be explored.

To highlight the complexity arising from a replanning formulation, let us assume that in our example there is a connection from location l3 to l4, and from l2 and l4. That is, the rover can move not only from l1 to l2, but also from l2 to l4 and from l4 to l3, for *all* the provided modalities. In addition, for simplicity reasons, assume that from that point (l3), the only possible sequence of actions toward the goal is given by a_2 and a_3.

While the reconfiguration mechanism can focus just on the impact on resources given by the selection of modalities for the next actions (TP, DRIVE, COMM), it is quite evident that a traditional replanner should deal with a larger search space. As matter of fact, it should consider also the (several) possible trajectories of states given by exploring the alternatives ways of reaching location l2 (drive(r1,l2,l4)), for all the possible modalities of execution. That is, it will have to cope with both the propositional and resource constraints of the arising planning problem. For a deeper discussion on this aspect, and on the computational complexity relation between the reconfiguration and the overall numeric planning problem see [28, 25].

As we will see in the next section, this different characterization is crucial for determining the performance of the reconfiguration over replanning from scratch, and even over the state of the art plan repair strategy presented in [10].

6 Experimental Validation

To assess the effectiveness of our proposal, we evaluated two main parameters: (1) the computational cost of reconfiguration, and (2) the competence of ReCon, that is, the ability of completing a mission.

To this aim, we have compared ReCon with three alternative strategies: REPLAN, LPG-ADAPT and NoRep. Whenever the plan becomes resources inconsistent, both REPLAN and LPG-ADAPT stop the execution of the plan and try to recover from the impasse. REPLAN searches a new plan completely from scratch, while LPG-ADAPT uses the old plan as a guidance to speed-up the resolution process[7]. Conversely, NoRep just stops the plan execution as soon as it is no longer valid. We used REPLAN and LPG-ADAPT to better assess the contribution of ReCon w.r.t. the current state of the art in (re)planning dealing with consumable resources.

We have implemented ReCon in Java 1.7 by exploiting the PPMaJaL library[8]; the Choco CSP solver (version 2.1.3)[9] has been used in the **Adapt** algorithm to find an alternative configuration. Concerning the REPLAN strategy, we invoke Metric-FF [17] by converting the rover actions with modalities in PDDL 2.1 actions. In order to study the effectiveness of the strategy in an on-line plan execution context, we allotted each computation with a time deadline; this parameter is critical for the competence of the system being tested. For this reason, we report results obtained with three different time deadlines: 5

[7]LPG-ADAPT, [15], is the plan adaptation extension of LPG, [16], one of the more awarded systems throughout the planning competitions of the last decade. LPG-ADAPT can be considered the state of the art in the context of plan adaptation.

[8]www.di.unito.it/ ∼scala

[9]The Choco Solver implements the state of the art algorithms for constraint programming and has already been used in space applications, see [6]. Choco can be downloaded at http://www.emn.fr/z-info/choco-solver/.

secs, 30 secs and 60 secs. Each time deadline corresponds to the maximum time that is given to the reconfiguration/replanning for providing a valid solution, once a plan becomes invalid throughout the whole execution process.

Our tests set consists of 168 plans; each plan involves up to 34 actions (i.e., drives, take pictures, and communications), it is fully instantiated (a modality has been assigned to each action), and feasible since all the goal constraints are satisfied when the plan execution starts.

To simulate unexpected deviations in the consumption of the resources, we have run[10] each test in thirteen different settings. In each of these settings we have noised the amount of resources consumed by the actions. In particular, in setting 1, an action consumes 10% more than expected at planning time. In setting 2, the noise was increased to 15%, and so on until in setting 13 where the noise was set to 70%, i.e. an action consumes 70% more resources than initially predicted.

On the left of Figures 5, 6 and 7 we report the competence - measured as the percentage of performed actions in the plan - of the three strategies, in the thirteen settings of noise we have considered. As expected, the competence decreases as long as the amount of noise increases, for all the strategies tested. ReCon resulted more competent than both REPLAN and LPG-ADAPT. Even though REPLAN and LPG-ADAPT can modify all the aspects of the plan structure, and hence they are theoretical more competent than ReCon, the search spaces generated by the overall arising planning problems turned out to be too large from the point of view of REPLAN and LPG-ADAPT. The timeouts are reached by a large number of cases in all cpu-time settings, and this is the reason of a lower competence of REPLAN and LPG-ADAPT. In particular we can observe a large gap between the percentage of plan completed by ReCon and REPLAN in all the cpu-time settings. In our experiments, also with an increased cpu-time at disposal, REPLAN was not able to find solutions for many cases. As refers the comparison with LPG-ADAPT, the gap is more limited for the high level of noise, showing how LPG-ADAPT can effectively takes advantage from the knowledge of the previous plan. It is worth noting that, as expected, this gap decreases as long as the noise increase; this is of course due to the contribution of the flexibility of the search space in which LPG-ADAPT and REPLAN can find a solution.

Observing the differences between the competence of the systems over the various cpu-time setting, it is clear that this parameter is crucial for the competence of LPG-ADAPT, while it does not affect the competence of ReCon, and neither of REPLAN. As expected, the LPG-ADAPT competence is quite competitive with ReCon for the 60 secs; in particular in the first 4 settings of noise, ReCon outperforms LPG-ADAPT, while with larger noises, LPG-ADAPT has more or less the same performances of ReCon. In the 5 secs setting both LPG-ADAPT and REPLAN are not competitive at all.

The right column of Figures 5, 6 and 7 reports the computational cost, on average, of the three strategies. Note that for each case considered, the compu-

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

tational cost corresponds to the sum of all the attempts to recovery from the failure (reconfiguration, plan-adaptation or replanning) performed until the end of the mission. Here the advantage of ReCon is very large in each experimental setup. In fact, even for the worst case (when the noise is set to be 70%), ReCon is extremely efficient, indeed it takes, on average, just 1,2 secs. Whereas, even for the cases with small amount of noise, as you can see in Figure 7, REPLAN takes about 7 secs of cpu-time till 50 secs employed for the worst cases, while LPG-ADAPT performs a little bit worse than REPLAN.

Finally, in Figure 8 we conclude by analyzing the number of invocations of the systems throughout the whole plan execution. Basically we collected the average number of attempts that the systems have performed whenever the plan turned out not valid during the execution. Observing the results, it is quite evident that the reconfiguration mechanism is invoked on average more times than the other architectures. This happens because, as long as the plan execution process goes on, the constraints becomes more and more tight, causing the detection mechanism to be invoked more frequently. Differently, each invocation of REPLAN generates a completely new plan; therefore the plan execution till the end is not directly related to the previous plan execution problem. This is the reason why REPLAN almost preserves the same amount of invocations throughout the cases we have tested. A similar trend can be found in comparing LPG-ADAPT with ReCon. Also LPG-ADAPT makes on the average less repair than ReCon; the difference of performances between REPLAN and LPG-ADAPT is probably due to the different way the underlying planning systems (LPG and Metric-FF) explores the search space. Of course this should be verified testing other numeric planners.

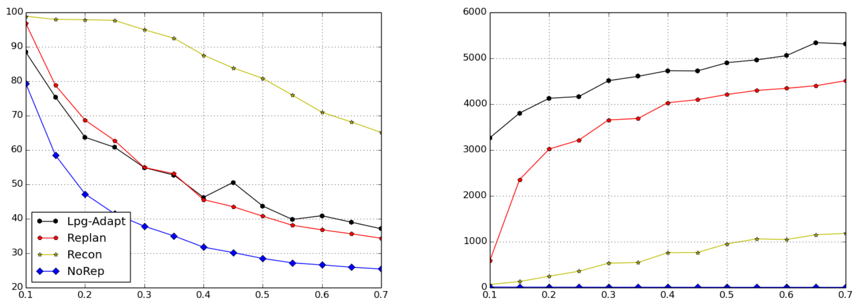

Figure 5: Competence (left) and Cpu-Time (right) - 5 secs setting

7 Conclusions

We have proposed in this paper a novel approach to the problem of robust plan execution. Rather than recovering from plan failures via a re-planning step (see

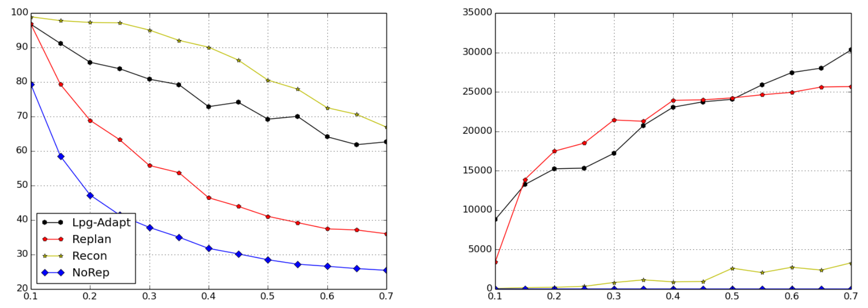

Figure 6: Competence (left) and Cpu-Time (right) - 30 secs setting

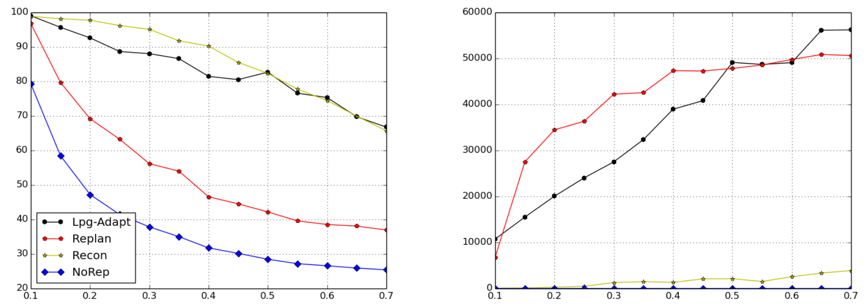

Figure 7: Competence (left) and Cpu-Time (right) - 60 secs setting

e.g., [14, 18, 13, 26]), we have proposed a methodology, called ReCon, based on the re-configuration of the plan actions. ReCon is justified in all those scenarios where a pure replanning approach is unfeasible. This is the case, for instance, of a planetary rover performing a space exploration mission. Albeit a rover must 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 mission plan without the approval from the ground control station.

ReCon presents many advantages w.r.t. re-planning. First of all, as the experiments have demonstrated, reconfiguring plan actions is computationally cheaper than synthesizing a new plan from scratch and even trying to adapt it via a classical plan adaptation tool (as the one reported in [15]). Moreover, ReCon leaves the high-level structure of the plan (i.e., the sequence of mission tasks) unchanged, but endows the rover with an appropriate level of autonomy for handling unexpected contingencies. ReCon can be considered as a complementary repair strategy to other works in the context of autonomy for space as those in [7]; as matter of facts, ReCon explores a different dimension of the

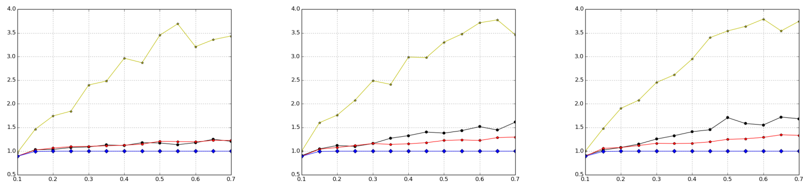

Figure 8: Average Number of Repairs over all the timeout settings

repair problem, which is based on an action-centered planning representation rather than on a timeline based perspective [12].

The solution described in this paper has been tested on a challenging domain such as a space exploration domain, but its applicability is not restricted to this domain. Many other robotic tasks could benefit of the proposed approach (combined with a generative approach, see [25]), since in many of them the need of adapting the plan execution to the resources constrains is very relevant.

The approach we have presented can be improved in a number of ways. A first important enhancement 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. Of course, the search for an optimal configuration is justified when the global constraints are not strict, and several alternative solutions are possible.

Acknowledgments

We would like to thank the Choco's team for making freely available the CSP solver, Joerg Hoffman for the Metric-FF planning system as well as Alfonso Gerevini, Alessandro Saetti and Ivan Serina for the LPG-ADAPT system.

References

[1] Barták, R., Ĉepek, O., Hejna, M.: Temporal reasoning in nested temporal networks with alternatives. In: Recent Advances in Constraints, Lecture Notes in Computer Science, vol. 5129, pp. 17–31. Springer Berlin Heidelberg (2008)

[2] Barták, R., Toropila, D.: Solving sequential planning problems via constraint satisfaction. Fundamamenta Informaticae 99(2), 125–145 (Apr 2010)

[3] Block, S.A., Wehowsky, A.F., Williams, B.C.: Robust execution of contingent, temporally flexible plans. In: Proc. of National Conference on Artificial Intelligence (AAAI-06): 802-808 (2006)

[4] Brenner, M., Nebel, B.: Continual planning and acting in dynamic multiagent environments. Journal of Autonomous Agents and Multiagent Systems 19(3), 297–331 (2009)

[5] Calisi, D., Iocchi, L., Nardi, D., Scalzo, C., Ziparo, V.A.: Context-based design of robotic systems. Robotics and Autonomous Systems (RAS) 56(11), 992–1003 (2008)

[6] Cesta, A., Fratini, S.: The timeline representation framework as a planning and scheduling software development environment. In: Proc. of P&S Special Interest Group Workshop (PLANSIG-10) (2009)

[7] Chien, S., Johnston, M., Frank, J., Giuliano, M., Kavelaars, A., Lenzen, C., Policella, N.: A generalized timeline representation, services, and interface for automating space mission operations. Tech. Rep. JPL TRS 1992+, Ames Research Center; Jet Propulsion Laboratory (June 2012)

[8] Conrad, P.R., Williams, B.C.: Drake: An efficient executive for temporal plans with choice. Journal of Artificial Intelligence Research 42, 607–659 (2011)

[9] desJardins, M., Durfee, E.H., Jr., C.L.O., Wolverton, M.: A survey of research in distributed, continual planning. AI Magazine 20(4), 13–22 (1999)

[10] Fox, M., Gerevini, A., Long, D., Serina, I.: Plan stability: Replanning versus plan repair. In: Proc. International Conference on Automated Planning and Scheduling (ICAPS-06). pp. 212–221 (2006)

[11] Fox, M., Long, D.: Pddl2.1: An extension to pddl for expressing temporal planning domains. Journal of Artificial Intelligence Research 20, 61–124 (2003)

[12] Fratini, S., Pecora, F., Cesta, A.: Unifying planning and scheduling as timelines in a component-based perspective. Archives of Control Sciences 18(2), 231–271 (2008)

[13] Garrido, A., C., G., Onaindia, E.: Anytime plan-adaptation for continuous planning. In: Proc. of P&S Special Interest Group Workshop (PLANSIG-10) (2010)

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

[15] Gerevini, A., Saetti, A., Serina, I.: Case-based planning for problems with real-valued fluents: Kernel functions for effective plan retrieval. In: Proc. of European Conference on AI (ECAI-12). pp. 348–353 (2012)

[16] Gerevini, A., Saetti, I., Serina, A.: An approach to efficient planning with numerical fluents and multi-criteria plan quality. Artificial Intelligence 172(8-9), 899–944 (2008)

[17] Hoffmann, J.: The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. Journal of Artificial Intelligence Research 20, 291–341 (2003)

[18] van der Krogt, R., de Weerdt, M.: Plan repair as an extension of planning. In: Proc. International Conference on Automated Planning and Scheduling (ICAPS-05). pp. 161–170 (2005)

[19] Lopez, A., Bacchus, F.: Generalizing graphplan by formulating planning as a csp. In: Proc. of International Conference on Artificial Intelligence (IJCAI-03). pp. 954–960 (2003)

[20] Micalizio, R.: Action failure recovery via model-based diagnosis and conformant planning. Computational Intelligence 29(2), 233–280 (2013)

[21] Micalizio, R., Scala, E., Torasso, P.: Intelligent supervision for robust plan execution. In: Lecture Notes in Computer Science, vol. 6954. pp. 151–163 (2011)

[22] Muscettola, N.: Hsts: Integrating planning and scheduling. Tech. Rep. CMU-RI-TR-93-05, Robotics Institute, Pittsburgh, PA (March 1993)

[23] Narendra, J., Rochart, G., Lorca, X.: Choco: an open source java constraint programming library. In: CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08). pp. 1–10 (2008)

[24] Policella, N., Cesta, A., Oddi, A., Smith, S.: Solve-and-robustify. Journal of Scheduling 12, 299–314 (2009)

[25] Scala, E., Micalizio, R., Torasso, P.: Robust plan execution via reconfiguration and replanning. AI Communications p. to appear (2014)

[26] Scala, E.: Numeric kernel for reasoning about plans involving numeric fluents. Lecture Notes in Computer Science, vol. 8249, pp. 263–275 (2013)

[27] Scala, E.: Numerical kernels for monitoring and repairing plans involving continuous and consumable resources. In: Proc. of International Conference on Agents and Artificial Intelligence (ICAART-13). pp. 531–534 (2013)

[28] Scala, E.: Reconfiguration and Replanning for robust Execution of Plans Involving Continous and Consumable Resources. Ph.D. thesis, Department of Computer Science - University of Turin (2013)